

# jQuery 风暴

—完美用户体验

张子秋  
编著



- 学好jQuery 1.4，更少代码，更炫效果
- 结合jQuery 1.4，快速掌握LBS地图开发
- 抢滩jQuery Mobile，改善移动互联网体验

# jQuery风暴

## —完美用户体验

---

· 张子秋 编著 ·

电子工业出版社·

Publishing House of Electronics Industry

北京·BEIJING

## 内 容 介 绍

本书全面讲解了 jQuery 的各种技术,包括基础特性、各类函数的介绍、使用 jQuery 进行 AJAX 调用、jQuery 插件的开发和使用等。书中对页面设计和开发人员影响深远的 jQuery UI 做了重点讲解,还穿插了部分 JavaScript 的精华知识,指出了各种脚本开发的错误方法和误区。接着给出自定义 jQuery 脚本框架和百度地图开发的案例,从而掌握高效率页面开发经验和 LBS (Location Based Service) 应用开发的思路。最后,本书还专门讲解了新鲜出炉的 jQuery Mobile,让我们在移动互联网开发中先人一步。通过本书的学习将改变传统的 JavaScript 开发方式,对于 Web 开发来说意义深远。本书从丰富的实践案例去讲解 jQuery 用户体验,从而极大提升用户的直观感受。

jQuery 的大版本已经更新到了 1.4,这一版本的 jQuery 有了大量的更新,比如对于函数重新进行了分类,推出了全新的 API 文档,这相当于重新整理了 jQuery 的知识体系结构。所以,本书的写作目的不仅仅是讲解 jQuery 中的技术知识,更侧重建立完整清晰的 jQuery 知识体系,让读者知道 jQuery 的骨架,以后也能够通过自己深入的学习让羽翼更加丰满。本书注重用户体验方面的介绍,在案例的选取上特别注重 B/S 企业的实践经验。

本书适合于 Web 开发工程师、用户界面设计师、前端架构师、用户体验设计师、移动互联网开发工程师、想深入学习 jQuery 知识的高级开发人员参考学习,还可作为高等院校相关专业的教学参考书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有,侵权必究。

### 图书在版编目(CIP)数据

jQuery 风暴:完美用户体验/张子秋编著.—北京:电子工业出版社,2011.4  
ISBN 978-7-121-12891-2

I. ①J… II. ①张… III. ①Java 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字(2011)第 016548 号

策划编辑:张春雨

责任编辑:许 艳

特约编辑:赵树刚

印 刷:北京东光印刷厂

装 订:三河市皇庄路通装订厂

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本:787×1092 1/16 印张:19.25 字数:492.8 千字 彩插:1

印 次:2011 年 4 月第 1 次印刷

印 数:4000 册 定价:39.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 zlts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010) 88258888。

# 前言

jQuery 无疑是现在最流行的脚本类库之一。jQuery 可以帮助用户轻松地将动态功能应用到网页，而且能最小化代码量。jQuery 有着丰富而绚丽的应用，包括访问部分网页、快速修改网页内容、添加动画、jQuery UI 控件的应用、应用 AJAX 技术、打造自己的脚本框架、开发热门的 LBS（Location Based Service）应用、jQuery Mobile 开发移动互联网服务等。

随着用户对网站体验的要求越来越高，在用户至上这一需求的推动下，作为开发者或者网站前台设计师而言，如果能在 B/S 的网站架构下真正用好 jQuery 来提高用户体验和满意度，那么他或许能够更加自信，在职场中也能把握更大的主动权。

## 这本书的特点有哪些？

如果没有透彻理解 JavaScript 的精髓，那么对于 jQuery 的深入应用将缺乏基石。因此本书特别安排了内容来讲解 JavaScript，内容涉及命名规范、function、this 指针、DOM、JSON 等。有了这些基础之后，我们开始踏上 jQuery 之旅，内容涉及如何利用选择器把效果应用到段落和列表、事件处理和对不同表元素执行验证的技术、应用视觉效果、导航、AJAX、jQuery UI 和打造属于自己的 jQuery 脚本框架。

随着 LBS（Location Based Service）的应用越来越广泛，本书还专门讲解了百度地图的开发。jQuery Mobile 给移动互联网的应用开发提供了便利性，本书在第一时间研究 jQuery Mobile，并把相关的心得呈现出来。

如果你对 JavaScript、HTML、CSS 和 jQuery 稍有了解，那么这本书正是为你而准备的。因为本书涵盖了利用 jQuery 展开工作时可能遇到的大多数问题，而且本书用通俗而浅显的文字、个性化的应用案例，给出了学习 jQuery 的范例，用来解释每一个新概念，并且每个范例都提供完整的代码。

本书对于知识的组织、整理、归类，以及如何让读者能够建立完整的 jQuery 知识体系进行了自己独特的探索。此外，jQuery 的大版本已经更新到了 1.4，这一版本的 jQuery 有了大量的更新，比如对于函数重新进行了分类，推出了全新的 API 文档，这相当于重新整理了 jQuery 的知识体系结构。所以，本书写作目的不仅仅是讲解 jQuery 中的技术知识，更侧重建立完整清晰的 jQuery 知识体系，让读者知道 jQuery 的骨架，以后也能够通过自己深入的学习让羽翼更加丰满。

## 这本书适合你吗？

- ◆ 听说过 jQuery，明白 jQuery 在用户体验方面很“给力”，迫切想掌握该框架的您。
- ◆ 了解 JavaScript、HTML、CSS，想让网站交互性更加完美的您。
- ◆ 接触过 jQuery，但是理解不是那么深刻，特别对 jQuery 1.4 新特性很着迷的您。
- ◆ 天天接触 jQuery，但是不晓得怎么来打造个性化 jQuery 脚本框架的您。
- ◆ 对 LBS（Location Based Service）已经如雷贯耳了，希望开发适合自己应用的您。
- ◆ 刚刚听说 jQuery Mobile 推出了，正在琢磨着如何改善移动互联网体验的您。

## 能学到什么？

- ◆ 使用 jQuery 的 API 文档的技巧。

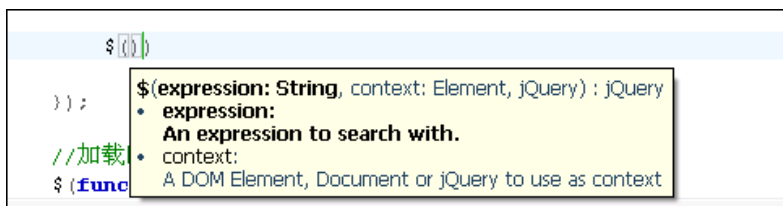




- ◆ 把效果应用到段落和列表。
- ◆ 灵活设置页面布局和页面导航。
- ◆ 丰富的事件处理机制和动画效果。
- ◆ jQuery UI 完美应用。
- ◆ 自定义和封装自己的 jQuery 框架。
- ◆ jQuery 与百度地图构建 LBS (Location Based Service) 热门应用。
- ◆ jQuery Mobile, 开发基于移动互联网平台的应用。

## 本书涉及哪些 jQuery 的新特性?

### 1. 完美的 JS 智能提示支持



### 2. 功能强大的选择器 (支持 CSS1-3 选择器及各种功能强大的选择器)



### 3. 性能最佳的选择器

selectors	MooTools 1.2	MooTools 1.3	jQuery 1.4.2	Prototype 1.6.0.2	YUI 2.5.2 Selector beta	Dojo 1.1.1
body	1 ms   1 found	1 ms   1 found	0 ms   1 found	0 ms   1 found	1 ms   1 found	1 ms   1 found
div	1 ms   51 found	0 ms   51 found	0 ms   51 found	0 ms   51 found	0 ms   51 found	1 ms   51 found
body div	1 ms   51 found	1 ms   51 found	0 ms   51 found	4 ms   51 found	0 ms   51 found	0 ms   51 found
div p	1 ms   140 found	1 ms   140 found	0 ms   140 found	3 ms   140 found	2 ms   140 found	0 ms   140 found
div > p	1 ms   134 found	1 ms   134 found	0 ms   134 found	2 ms   134 found	1 ms   134 found	3 ms   134 found
div ~ p	1 ms   22 found	0 ms   22 found	0 ms   22 found	11 ms   22 found	2 ms   22 found	1 ms   22 found
div ~ p	2 ms   183 found	1 ms   183 found	0 ms   183 found	6 ms   183 found	3 ms   183 found	2 ms   183 found
div[class~="area"][class~="single"]	1 ms   43 found	0 ms   43 found	0 ms   43 found	3 ms   43 found	0 ms   43 found	0 ms   43 found
div p a	0 ms   12 found	0 ms   12 found	0 ms   12 found	3 ms   12 found	3 ms   12 found	1 ms   12 found
*****	*****	*****	*****	*****	*****	*****
final time (less is better)	47	26	5	179	130	26

测试结果使用 win7+Chrome 浏览器 省略部分详细信息 访问 <http://mootools.net/slickspeed/> 可以获得每个人自己机器的性能测试结果

### 4. 支持多种浏览器

IE 6.0+、FF 2+、Safari 3.0+、Opera 9.0+、Chrome。

### 5. 优雅的链式语法

```
$(".myClass").css("color", "red").show();
```

### 6. AJAX 操作

```
$("#divResult").load("../data/AjaxGetCityInfo.aspx", { "resultType": "html" });
```





## 7. Datpicker 日历控件

**机票搜索**

单程 往返

出发城市:

到达城市:

出发时间: 2010-08-15

搜索

2010年 八月							2010年 九月							2010年 十月						
一	二	三	四	五	六	日	一	二	三	四	五	六	日	一	二	三	四	五	六	日
						1			1	2	3	4	5					1	2	3
2	3	4	5	6	7	8	6	7	8	9	10	11	12	4	5	6	7	8	9	10
9	10	11	12	13	14	15	13	14	15	16	17	18	19	11	12	13	14	15	16	17
16	17	18	19	20	21	22	20	21	22	23	24	25	26	18	19	20	21	22	23	24
23	24	25	26	27	28	29	27	28	29	30				25	26	27	28	29	30	31
30	31																			

## 8. Dialog 对话框控件

**Demo. 每个弹出层内容不同, 弹出层内容存储**

红色 绿色

**Demo. 弹出iFrame**

显示弹出层

**iFrame弹出层**

**会员登录**

帐号:

登录名不能为空

密码:  [找回密码及卡号?](#)

密码不能为空

登录并继续

**非会员注册**

[注册成为艺龙会员](#)

## 9. Tab 控件

One Two **Three**

Tab3内容 鼠标悬浮时切换

## 10. Accordion 手风琴菜单控件

菜单1

菜单2 

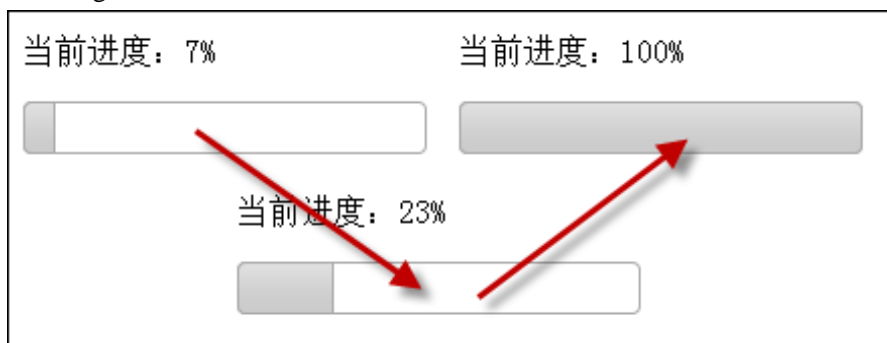
菜单2的内容

菜单3

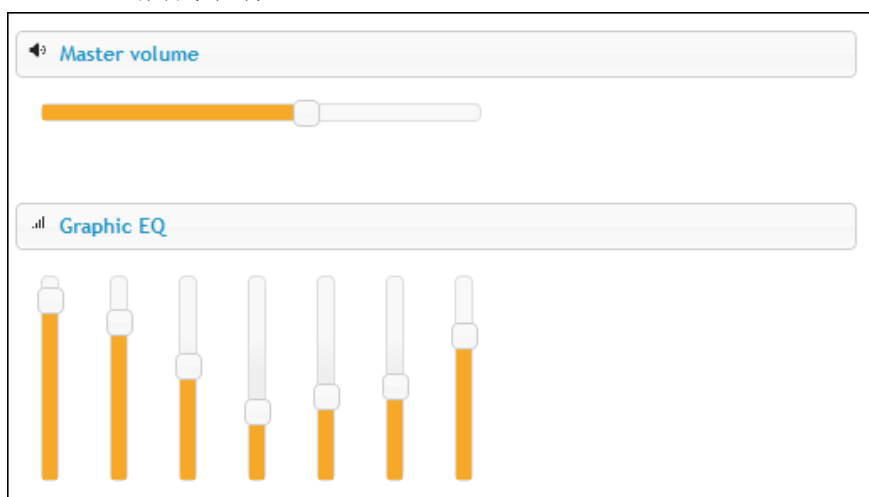




### 11. Progressbar 进度条控件



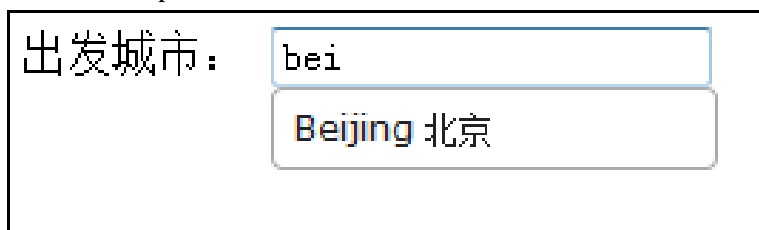
### 12. Slider 滑动条控件



### 13. Button 按钮控件



### 14. Autocomplete 自动提示控件





## 15. 易于扩展（基于 jQuery 开发的日历框）

城市：

入住日期：

离店日期：

价格范围：

2010年7月2010年8月

酒店名称：

日	一	二	三	四	五	六	日	一	二	三	四	五	六
					1	2	3	1	2	3	4	5	6
4	5	6	7	8	9	10	8	9	10	11	12	13	14
11	12	13	14	15	16	17	15	16	17	18	19	20	21
18	19	20	21	22	23	24	22	23	24	25	26	27	28
25	26	27	28	29	30	31	29	30	31				

搜索

## 16. jQuery Mobile（开发基于移动互联网平台的炫丽应用）

本地搜索

搜索地点：

在周围搜索：

搜索



## 致谢

写作的过程漫长而且艰辛，历时一年多，甚至跨越了 jQuery 版本（在刚开始写作时还是 1.3.2 版本）。

在写作本书的时候，我脑子里总是不停地思考，如何把知识讲解得简单，如何让读者更容易读懂，阅读起来更加流畅。因为我自己就常常遇到看书时，发现书中的知识过于跳跃或者语言晦涩很难理解的情况。但是为了易于读者理解本书的内容，本书也牺牲了一些展示复杂代码和技术的机会。如果你一口气看完全书后，有跃跃欲试的感觉，甚至感觉还不过瘾的话，那么说明你上道了，这总比看了一半郁闷地扔掉本书要好多了。

本书由张子秋编著。在编写过程中，万雷、王斌、张强林、张赛桥、黄北军、王文佳、王晓也做了大量的工作。成都道然科技有限责任公司参与了本书审核校对工作。此外，还





要感谢艺龙网、时光网、我的母亲周立波、我的父亲张跃、我的领导王雪、于泳洋、邹正宸。由于笔者能力和知识有限，如有疏漏，还请读者批评指正，E-mail: sharepub@126.com。

## 本书代码与支持

为了低碳生活，本书没有配光盘，而是采用网上下载代码的方式。本书的源代码下载，以及有疑问想询问作者，可以访问作者的博客：<http://www.cnblogs.com/zhangziqu>。

更多的资料也可以从 <http://www.dozan.cn> 网站上获取。



# 目 录

第 1 章 jQuery 入门 .....	1
1.1 认识 jQuery .....	2
1.1.1 认识 jQuery .....	2
1.1.2 jQuery 之美 .....	3
1.1.3 jQuery 与其他脚本类库的比较 .....	5
1.2 上手 jQuery .....	6
1.2.1 jQuery 版本介绍 .....	6
1.2.2 在 Visual Studio 中使用 jQuery .....	7
1.2.3 在 Aptana 中使用 jQuery .....	8
1.3 Hello jQuery 实例 .....	10
1.3.1 添加脚本引用 .....	10
1.3.2 添加 DOM 元素 .....	11
1.3.3 创建页面脚本对象 .....	11
1.4 小结 .....	12
第 2 章 必须知道的 JavaScript 知识 .....	13
2.1 JavaScript 基础 .....	14
2.1.1 JavaScript 与 ECMAScript .....	14
2.1.2 JavaScript 中的值类型和引用类型 .....	14
2.1.3 JavaScript 中的原始类型 .....	15
2.1.4 undefined、null 和 typeof 运算符 .....	15
2.1.5 变量声明 .....	17
2.1.6 JavaScript 命名规范 .....	17
2.1.7 变量的作用域与闭包 .....	18
2.2 悟透 JavaScript 中的 function .....	20
2.2.1 使用 function 声明方法和类型 .....	20
2.2.2 function 的本质 .....	21
2.2.3 new 运算符 .....	21
2.2.4 function 的 arguments 参数对象 .....	22
2.2.5 理解 this 指针 .....	22
2.3 JavaScript 中的原型 .....	24
2.3.1 使用原型实现 JavaScript 的面向对象 .....	24
2.3.2 使用原型链实现继承 .....	25
2.4 DOM .....	26
2.4.1 什么是 DOM .....	26
2.4.2 操作 HTML DOM 对象 .....	26
2.4.3 DOM 元素与 HTML 元素 .....	28





2.5 其他 JavaScript 秘籍.....	29
2.5.1 数据通信格式 JSON.....	29
2.5.2 动态语言——eval.....	31
2.5.3 JavaScript 中的逻辑运算符.....	32
2.6 小结.....	35
第 3 章 jQuery 核心基础.....	36
3.1 jQuery 对象.....	37
3.1.1 什么是 jQuery 对象.....	37
3.1.2 jQuery 对象深入解析.....	37
3.1.3 jQuery 对象转换为 DOM 对象.....	38
3.1.4 DOM 对象转化为 jQuery 对象.....	39
3.1.5 jQuery 对象的链式操作.....	39
3.1.6 “\$”变量的使用.....	40
3.1.7 解决多类库冲突——“\$”变量冲突问题.....	41
3.2 jQuery 文档处理程序.....	42
3.2.1 jQuery 文档处理程序介绍.....	42
3.2.2 文档处理程序的优势.....	43
3.2.3 jQuery 文档处理程序深入解析.....	44
3.2.4 jQuery 文档处理程序注意事项.....	46
3.3 jQuery 帮助文档.....	49
3.3.1 jQuery API 在线帮助文档.....	49
3.3.2 jQuery API 在线帮助文档分类.....	50
3.3.3 jQuery API 中文帮助文档.....	51
3.4 小结.....	52
第 4 章 万能的 jQuery 选择器.....	53
4.1 jQuery 选择器基础.....	54
4.1.1 什么是 jQuery 选择器.....	54
4.1.2 jQuery 选择器核心函数.....	54
4.1.3 jQuery 选择器分类.....	55
4.1.4 使用 jQuery 选择器实验室.....	55
4.1.5 选择器使用技巧.....	57
4.2 基础选择器.....	59
4.2.1 基础选择器列表.....	59
4.2.2 基础选择器使用要点.....	59
4.3 层次选择器.....	59
4.3.1 层次选择器列表.....	60
4.3.2 层次选择器使用要点.....	60
4.4 基本过滤器.....	61
4.4.1 基本过滤器列表.....	61





4.4.2 基本过滤器使用要点.....	62
4.5 内容过滤器 .....	63
4.5.1 内容过滤器列表.....	63
4.5.2 内容过滤器使用要点.....	63
4.6 可见性过滤器 .....	63
4.6.1 可见性过滤器列表.....	63
4.6.2 可见性过滤器使用要点.....	64
4.7 属性过滤器 .....	64
4.7.1 属性过滤器列表.....	64
4.7.2 属性过滤器使用要点.....	65
4.8 子元素过滤器 .....	65
4.8.1 子元素过滤器列表.....	65
4.8.2 子元素过滤器使用要点.....	66
4.9 表单类别过滤器 .....	66
4.9.1 表单类别过滤器列表.....	66
4.9.2 表单类别过滤器使用要点.....	67
4.10 表单属性过滤器 .....	68
4.10.1 表单属性过滤器列表.....	68
4.10.2 表单属性过滤器使用要点.....	68
4.11 小结 .....	69
第 5 章 管理 jQuery 对象集合 .....	70
5.1 动态创建元素 .....	71
5.1.1 使用 JavaScript 创建对象 .....	71
5.1.2 使用 jQuery 创建对象 .....	72
5.1.3 创建对象常见错误.....	73
5.2 过滤函数——筛选对象集合 .....	74
5.2.1 过滤函数列表 .....	74
5.2.2 过滤函数要点 .....	74
5.3 查找函数——找到目标对象 .....	76
5.3.1 查找函数列表 .....	77
5.3.2 查找函数要点 .....	78
5.4 串联函数——操作对象链 .....	80
5.4.1 串联函数列表 .....	80
5.4.2 串联函数要点 .....	80
5.5 小结 .....	82
第 6 章 使用 jQuery 操作元素 .....	83
6.1 DOM 属性与 HTML 元素属性.....	84
6.1.1 区分 DOM 属性与 HTML 元素属性 .....	84
6.1.2 使用 JavaScript 操作 DOM 属性.....	85
6.1.3 使用 JavaScript 操作 HTML 元素属性.....	86





6.2 使用 jQuery 操作 DOM.....	87
6.2.1 使用 jQuery 操作元素属性.....	88
6.2.2 使用 jQuery 操作元素 CSS .....	92
6.2.3 偏移量 offset 分类函数.....	96
6.2.4 用于测量的 Dimensions 分类函数.....	98
6.2.5 使用 jQuery 改变元素内容.....	100
6.3 小结 .....	102
第 7 章 事件与事件对象.....	103
7.1 DOM 事件模型 .....	104
7.1.1 DOM 事件流 .....	104
7.1.2 事件处理函数 .....	106
7.1.3 事件对象 .....	107
7.2 jQuery 事件模型 .....	108
7.2.1 jQuery 中的事件流 .....	108
7.2.2 jQuery 事件绑定函数 .....	110
7.2.3 事件处理函数中的 this 指针 .....	115
7.2.4 jQuery 事件对象 .....	116
7.3 jQuery 特殊事件 .....	122
7.3.1 对象监听函数 live 和 die.....	122
7.3.2 改进的鼠标事件 mouseenter、mouseleave 和 hover .....	124
7.3.3 改进的焦点事件 focusin 和 focusout .....	126
7.4 小结 .....	127
第 8 章 使用 AJAX 增加用户体验 .....	128
8.1 原始 AJAX 与 jQuery 中的 AJAX.....	129
8.1.1 原始 AJAX 应用举例 .....	129
8.1.2 jQuery 中的 AJAX 快餐 .....	130
8.2 使用 jQuery 的 AJAX 函数进行页面交互 .....	130
8.2.1 AJAX 快捷函数 .....	131
8.2.2 底层函数 ajax()和 ajaxSetup().....	136
8.2.3 AJAX 帮助函数 .....	141
8.2.4 AJAX 全局事件 .....	145
8.3 跨域的 AJAX-JSONP .....	146
8.3.1 什么是 JSONP.....	146
8.3.2 JSONP 实现原理.....	148
8.3.3 JSONP 在 jQuery 中的应用.....	148
8.4 小结 .....	149
第 9 章 jQuery 动画——让页面动起来.....	150
9.1 jQuery 动画基础 .....	151
9.1.1 动画入门实例 .....	151





9.1.2	jQuery 动画分类 .....	152
9.1.3	jQuery 动画实验室 .....	152
9.1.4	jQuery 动画时间参数 .....	152
9.1.5	jQuery 动画回调函数 .....	153
9.2	基础动画函数 .....	153
9.2.1	基础动画实例 .....	154
9.2.2	基础动画详解 .....	154
9.3	渐变动画函数 .....	155
9.3.1	渐变动画实例 .....	156
9.3.2	渐变动画详解 .....	156
9.4	滑动动画函数 .....	157
9.4.1	滑动动画实例 .....	158
9.4.2	滑动动画详解 .....	158
9.5	自定义动画函数 .....	158
9.5.1	jQuery 队列 .....	159
9.5.2	动画全局开关 .....	161
9.5.3	停止元素动画 .....	162
9.5.4	自定义动画效果.....	164
9.6	小结 .....	168
第 10 章	jQuery 工具函数 .....	169
10.1	jQuery 工具函数基础 .....	170
10.1.1	工具函数说明.....	170
10.1.2	jQuery 工具函数概览 .....	170
10.2	浏览器特性检测 .....	171
10.2.1	浏览器特性检测的演变.....	171
10.2.2	检测浏览器类型和版本.....	172
10.2.3	浏览器特性检测.....	174
10.3	数组和对象操作 .....	176
10.3.1	遍历数组和对象.....	178
10.3.2	过滤数组 .....	178
10.3.3	数组和对象合并.....	179
10.3.4	数组和对象转换.....	181
10.3.5	排序和过滤 DOM 元素集合 .....	182
10.3.6	转换 JSON 字符串 .....	183
10.4	其他工具函数 .....	184
10.4.1	字符串 trim 操作 .....	184
10.4.2	判断函数 .....	184
10.4.3	jQuery 中的全局 eval 函数.....	185
10.4.4	制造一个空函数.....	185
10.4.5	检查结点包含关系.....	186





10.4.6	修改函数上下文.....	186
10.4.7	jQuery 中的队列函数 .....	187
10.5	扩展 jQuery 工具函数 .....	187
10.5.1	使用 JavaScript 扩展工具函数 .....	187
10.5.2	使用 jQuery.extend() 函数扩展工具函数.....	188
10.6	小结 .....	188
第 11 章	拿来主义——jQuery 插件 .....	189
11.1	jQuery 插件基础 .....	190
11.1.1	jQuery 插件介绍.....	190
11.1.2	区别 jQuery 插件与工具函数.....	190
11.1.3	寻找合适的 jQuery 插件.....	191
11.1.4	合理使用 jQuery 插件.....	191
11.2	jQuery 插件开发 .....	191
11.2.1	为插件起一个名字.....	192
11.2.2	编写结构代码.....	192
11.2.3	设计插件参数.....	192
11.2.4	使用插件.....	193
11.2.5	插件开发要点.....	193
11.3	实战表单验证插件 .....	193
11.3.1	应用实例.....	194
11.3.2	验证方法.....	196
11.3.3	验证消息.....	196
11.3.4	验证规则.....	197
11.3.5	表单提交.....	198
11.3.6	DEBUG 模式.....	198
11.3.7	多表单验证.....	198
11.4	实战自动完成插件 .....	198
11.4.1	应用实例.....	199
11.4.2	准备数据源.....	201
11.4.3	设置关键函数.....	201
11.4.4	为控件添加 Result 事件函数.....	202
11.4.5	匹配中文.....	203
11.4.6	其他注意事项.....	203
11.5	小结 .....	203
第 12 章	页面的华丽外衣——jQuery UI .....	204
12.1	jQuery UI 基础.....	205
12.1.1	jQuery UI 简介 .....	205
12.1.2	jQuery UI 分类 .....	206
12.2	Datepicker 日历控件.....	207





12.2.1	应用实例 .....	207
12.2.2	日历框参数 .....	211
12.2.3	日历框事件 .....	214
12.2.4	日历框方法 .....	215
12.3	Dialog 对话框控件 .....	216
12.3.1	对话框应用场景.....	216
12.3.2	应用实例 .....	217
12.3.3	计算对话框位置.....	221
12.3.4	取消冒泡和浏览器默认行为.....	221
12.3.5	设置动画效果与取消动画.....	221
12.3.6	动态提示类对话框的数据传递.....	222
12.3.7	更换主题 .....	222
12.4	TAB 标签控件.....	222
12.4.1	应用实例 .....	222
12.4.2	注意 HTML 结构 .....	224
12.4.3	活用事件 .....	225
12.5	Accordion 手风琴菜单控件.....	226
12.5.1	应用实例 .....	226
12.5.2	关键点讲解 .....	229
12.6	Progressbar 进度条控件 .....	230
12.6.1	应用实例 .....	230
12.6.2	实例讲解 .....	231
12.7	Slider 滑动条控件.....	232
12.7.1	应用实例 .....	232
12.7.2	实例讲解 .....	234
12.8	button 按钮控件 .....	234
12.8.1	应用实例 .....	234
12.8.2	实例讲解 .....	237
12.9	autocomplete 自动提示控件.....	238
12.9.1	应用实例 .....	238
12.9.2	实例讲解 .....	239
12.10	小结 .....	241
第 13 章	基于 jQuery 打造脚本框架 .....	242
13.1	页面脚本管理 .....	243
13.1.1	使用面向对象的方式管理页面脚本.....	243
13.1.2	页面脚本事件.....	245
13.1.3	切割脚本文件.....	247
13.1.4	为脚本文件添加智能提示.....	248
13.1.5	合并及压缩脚本文件.....	249
13.2	公共脚本类库 .....	250







13.2.1	template 模板方法.....	250
13.2.2	修改函数上下文的方法.....	250
13.2.3	反序列化 unparam()方法.....	251
13.2.4	操作 Cookie 的方法.....	252
13.2.5	JSON 转换方法.....	255
13.3	打造 jQuery UI 控件库.....	255
13.3.1	使用 jQuery UI.....	255
13.3.2	自定义日历控件.....	258
13.4	小结.....	263
第 14 章	jQuery 与百度地图实战.....	264
14.1	网站规划.....	265
14.1.1	网站主题.....	265
14.1.2	用户人群.....	265
14.1.3	盈利模式.....	265
14.1.4	未来规划.....	265
14.2	网站实现.....	266
14.2.1	定义页面结构.....	266
14.2.2	实现样式.....	267
14.2.3	实现页面功能.....	268
14.2.4	页面重构.....	268
14.3	脚本详解.....	269
14.3.1	百度地图 API 介绍.....	269
14.3.2	使用百度地图 API.....	270
14.3.3	使用页面脚本框架.....	271
14.3.4	使用 jQuery UI.....	272
14.3.5	本地搜索.....	273
14.3.6	公交和驾车搜索.....	275
14.4	小结.....	277
第 15 章	移动脚本框架 jQuery Mobile.....	278
15.1	jQuery Mobile 介绍.....	279
15.1.1	jQuery Mobile 的目的.....	279
15.1.2	jQuery Mobile 浏览器兼容性.....	279
15.1.3	jQuery Mobile 特性.....	280
15.2	jQuery Mobile 入门.....	280
15.2.1	Hello Mobile 实例.....	281
15.2.2	API 分类.....	282
15.2.3	页面结构.....	283
15.2.4	配置系统.....	285
15.2.5	事件处理.....	286





15.3	jQuery Mobile 与百度地图 API 综合实例 .....	287
15.3.1	实例效果 .....	287
15.3.2	定制页脚 .....	288
15.3.3	组织页面脚本.....	289
15.3.4	添加事件 .....	289
15.4	小结 .....	290





## 第 1 章



# jQuery 入门

本章带领大家进入 jQuery 的精彩世界，主要介绍 jQuery 的特性和其他类库的比较等，通过本章的学习可以对 jQuery 有一个全面的认识。此外，Hello jQuery 实例可以快速上手 jQuery，轻松开始 jQuery 之旅！

## 1.1 认识 jQuery

随着互联网的迅速发展, JavaScript 已经成为 Web 开发人员一定要使用的客户端脚本语言。比如为了提高用户体验使用的 AJAX 技术就是使用 JavaScript。会写 JavaScript 不难, 但是写好的人却不多, 由于 JavaScript 是动态语言, 极大的灵活性导致了项目中每个人截然不同的代码风格。而且现在的页面对于用户体验方面的要求越来越高, 酷炫的页面效果不仅仅吸引眼球, 更能够方便用户的交互。加上现在各种浏览器对于 JavaScript 的实现标准和解析方式不同, 所有这些都是对 Web 开发和设计人员的挑战。于是当互联网世界陷入灾难之时, 上天派出了 jQuery 拯救苍生。

### 1.1.1 认识 jQuery

jQuery 是一套轻量级的 JavaScript 类库, JavaScript 语言是在 Web 页面上使用的客户端脚本语言, 使用 jQuery 可以帮助我们迅速地完成任务, 并且实现的效果都是跨浏览器兼容的。毫不夸张地讲, jQuery 改变了我们的 JavaScript 编程方式, 正如 jQuery 的口号所说的“write less, do more!”(事半功倍)。

首先需要注意 jQuery 是脚本库而不是脚本框架, jQuery 提供的是核心函数, 用户可以在这些核心函数上开发自己的功能。jQuery 不会像脚本框架一样组织所有的脚本文件, 所以说 jQuery 是轻量级的。但是正因为轻巧, 所以用起来更得心应手。

另外 jQueryUI 的发布改变了 jQuery 插件繁多各自为战的情况, 可以说 jQueryUI 是用于用户交互的综合 jQuery 插件, 提供了丰富的模板皮肤和各种 UI 组件, 并且侵入性低, 对于提升用户体验有很大的帮助。

jQuery 是由 John Resig 创建的, 已经从 2006 年的 1.0 版本发展到 2009 年的 1.3.X 版本, 下面简单回顾 jQuery 的成长历史:

- ❑ jQuery 1.0 于 2006 年 8 月发布。作为第一个发布版本已经包含了 CSS 选择器、事件处理和 AJAX 接口函数。
- ❑ jQuery 1.1 于 2007 年 1 月发布。对于 API 做了大量的整理, 将不常用的函数进行合并或删除。
- ❑ jQuery 1.1.3 于 2007 年 7 月发布。大幅度改善了选择器的性能, 性能已经可以和 Prototype、MooTools、Dojo 等相媲美。
- ❑ jQuery 1.2 于 2007 年 12 月发布。取消了 XPath 选择器, 因为 CSS 选择器已经足够强大, 对函数的易用性和插件开发都有所改进。
- ❑ jQueryUI 于 2007 年 12 月发布。jQueryUI 的发布为实现丰富的用户界面和用户体验提供了强有力的基础, 后面的章节中也会重点介绍 jQueryUI 的使用。
- ❑ jQuery 1.2.6 于 2008 年 5 月发布。主要的更新在于性能的提升, 并且整合了 DimensionsPlugin 插件。
- ❑ jQuery 1.3 于 2009 年 1 月发布。它更新了 Sizzle 选择器引擎, 提高了很多函数方法的性能, 一举让 jQuery 成为最快的脚本类库, 此外还添加了 live 等事件委托函数。
- ❑ jQuery 1.4 在 2010 年 1 月 14 日, jQuery 的四岁生日时发布。该版本显著提高了最

常用的 jQuery 方法的性能,并且修复了非常多的 bug。jQuery 开发团队在开发 jQuery 1.4 时大幅增加了测试用例。jQuery 测试在所有主流浏览器 (Safari 3.2、Safari 4、Firefox 2、Firefox 3、Firefox 3.5、IE 6、IE 7、IE 8、Opera 10.10 以及 Chrome) 中全部通过。同时 1.4 版本还完全更换了 jQuery 的在线文档手册。启用了新的 `api.jquery.com` 作为手册的地址,此手册对于函数的组织和分类更加系统化。

通过自身不断的完善和更新, jQuery 已经应用得越来越广泛,到目前为止是使用最广泛的 JavaScript 类库之一,被微软和诺基亚等各大公司采用。

微软已经将 jQuery 作为御用脚本类库,它的各种 JavaScript 应用均以 jQuery 为基础,比如 ASP.NET Ajax。创建一个 ASP.NET MVC 项目时将默认引入 jQuery 类库。jQuery 与 Visual Studio 的配合更是完美,可以在 Visual Studio 中启用对 jQuery 类库函数的智能提示,大大提升了开发效率。

诺基亚使用 jQuery 帮助开发基于手机的 Web 运行平台,比如用 jQuery 开发新版本的手机地图等。

只要是需要使用 JavaScript 的地方, jQuery 就能发挥重要的作用。在富互联网时代,作为一名网站开发人员,如不懂 jQuery 都不好意思和别人打招呼!

### 1.1.2 jQuery 之美

jQuery 带给我们的是 JavaScript 开发的革命。在没有脚本库的日子,开发脚本绝对是一件容易出错的工作,并且实现类似跨浏览器、页面动画效果等都属于高难度的应用,“救世主” jQuery 有如下神力。

#### 1. 提供了功能强大的脚本函数类库

使用这些功能函数,能够帮助 Web 开发人员快速完成各种功能,而且会让代码异常简洁。比如,通过 jQuery 选择器一次选中页面上所有的 Checkbox 元素,通过 `each` 遍历选中的所有元素。

```
$("*:checkbox").each(function() { });
```

#### 2. 解决浏览器兼容性问题

JavaScript 脚本在不同浏览器的兼容性问题一直是 Web 开发人员的噩梦,常常一个页面在 IE 7、Firefox 下运行正常,但在 IE 6 下就出现莫名其妙的问题。针对不同的浏览器编写不同的脚本是一件痛苦的事情。jQuery 将开发人员从这个噩梦中拯救出来,比如 jQuery 中的 `event` 事件对象已经被格式化成所有浏览器通用,从前要根据 `event` 获取事件触发者,在 IE 下是 `event.srcElements` 而在 Firefox 等标准浏览器下是 `event.target`。jQuery 则通过统一 `event` 对象,可以在所有浏览器中使用 `event.target` 获取事件对象。当单击一个 `<a>` 元素时,默认会链接到 `href` 属性中的地址,使用 `event.preventDefault()` 方法就可以取消此行为 (浏览器默认行为)。

#### 3. 实现丰富的 UI

jQuery 可以实现比如渐变弹出、图层移动等动画效果,这样可获得更好的用户体验。以渐变效果为例,从前为了写一个可以兼容 IE 和 Firefox 的渐变动画,要使用大量 JavaScript



代码实现。费心费力不说，写完后没有太多帮助，过一段时间就忘记了，再开发类似的功能还要再次费心费力。现在使用 jQuery 就可以快速完成此类应用。

jQuery UI 就是用来帮助完成上面所有功能的 jQuery 插件库，除了实现效果外，它还提供了很多精美的主题和模板，可以实现在不改变任何代码的情况下更改样式风格。如图 1-1 所示为不同主题下的日历控件。



图 1-1 不同主题的日历控件

图 1-2 和图 1-3 是使用 dialog 组件实现的对话框拖拽和拉伸效果。



图 1-2 可拖拽



图 1-3 可缩放

#### 4. 优雅的链式编程方式

jQuery 中，常常使用如下所示的链式方式编程：

```
$("#myDiv").attr("title","myTitle").css("color","red");
```

上面的代码，首先获取到 id 为 myDiv 的元素，然后首先修改 title 属性，接着修改了 CSS 样式设置颜色为红色。jQuery 这种链式编程方式十分优雅，而且更适合用来描述语意。在最新版本的 Prototype 中也已经开始借鉴 jQuery 的这种语法结构，但是就导致了 Prototype 有两套编码方法。而 jQuery 作为创造者，使所有使用 jQuery 的编码人员都能用最优雅的方式编写代码。

#### 5. 学习简单，上手快

jQuery 的学习成本非常低，因为 jQuery 是轻量级、灵活的脚本框架，并且其官方提供了丰富的在线文档和实例，任何人都可以轻松上手。

#### 6. 太多了！等待我们——去发现





## 1.1.3 jQuery 与其他脚本类库的比较

jQuery 并不是唯一的 JavaScript 库，除了 jQuery 还有很多优秀的 JavaScript 库，比如 Prototype、Dojo，Ext、YUI、MooTools 等。每款 JavaScript 库都有其自身的优点和缺点，要根据不同的使用场景进行选择。如表 1-1 所示的是几款流行的框架比较。

表 1-1 脚本类库比较

类 库	jQuery & jQueryUI	Prototype & script.aculo.us	Dojo	ExtJS	YUI	MooTools
大小（可能随版本变化）	54KB	46-278KB	核 心 库 26KB, 可扩展	84-502KB	核 心 库 31KB, 可扩展	65KB
许可	MIT/GPL	MIT	BSD&AFL	Commercial & GPL	BSD	MIT
XMLHttpRequest 数据获取	是	是	是	是	是	是
JSON	是	是	是	是	是	是
拖放	是	是	是	是	是	是
简单视觉效果	是	是	是	是	是	是
动画/高级视觉效果	是	是	是	是	是	是
事件处理	是	是	是	是	是	是
页面后退/浏览历史管理	是	插件支持	是	依 靠 YUI History Manager	是	插件支持
输入验证	插件支持 (Validator)	是	是	是	是	插件支持
数据网络	插 件 支 持 (jgGrid/Ingrid/Flexgrid)	插件支持	是	是	是	插件支持
文本编辑器	插 件 支 持 (jwysiwyg/htmlbox/WYMeditor)	插件支持	是	是	是	插件支持
自动完成	插件支持 (AutoComplete)	是	是	插件支持	是	插件支持
HTML 自动生成	是	是	是	是	是	是
主题/皮肤更换	是	否	是	是	是	是
易用性/平稳退化	是	否	是	否	是	是
离线存储	否	否	是	Google Gears/Adobe Air	是	否
浏览器支持						
IE	6+	6+	6+	6+	6+	6+
Firefox	2+	1.5+	1.5+	1.5+	2+	1.5+
Safari	2+	2+	3+	3+	3+	2+
Opera	9+	9.25+	9+	9+	9+	9+





注：因为类库都还在不断地发展，上表的内容也许存在偏差。某些类库的功能空白，也许已开发完成或者已有相应的插件支持，所以此表格仅供参考。

如果仅看表格会发现 Dojo 和 YUI 是原生类库功能最强大的，尤其是 Dojo 还有 IBM、Sun 和 EBA 等大公司支持，功能可谓强大。包括笔者在内的很多人在最初选择脚本类库的时候也被其强大的功能所迷惑，选择了 Dojo。但是 Dojo 目前不能算是一个成熟的脚本类库，尤其是不适合做网站的开发。网站（WebSite）和 Web 应用程序（Web App）最大的不同就是使用者，网站面向的是互联网用户，而 Web 应用程序通常指内部系统（如管理系统）或者特定的使用人群（如 Google Docs 面向办公人员）。结果就是在开始尝到了 Dojo 类库的甜头后，发现 Dojo 并不适合网站开发，对于网站开发来说 Dojo 太重了，就像全副武装的陆战队员在进行百米赛跑。而且 Dojo 本身的确存在着尚未修复的缺陷，比如无法跨域加载类库文件（最新开发版本已经修复，官方版本尚未发布）。

同样重量级的还有 ExtJS 和 YUI。ExtJS 最早叫做 YUI-Ext，其实 ExtJS 是由于 YUI 的缺陷无法得到快速修复而另起门户的。由此可见 YUI 类库的显著问题就是更新缓慢，因为是官方的团队开发，它的优点是代码风格严谨。

对于互联网应用而言，轻量级的框架 jQuery 和 MooTools 都是很好的选择。MooTools 是开源社区形式下发展起来的 JS 框架。笔者认为有别于 jQuery 最明显的特点就是使用方式：MooTools 是完全地面向对象结构的脚本类库，在其代码组织风格、有无侵入等 Web 思想的理解，各个方面都呈现出少年新贵、武林新秀的姿态来。虽然目前的功能不是最强大的，但是都可以通过后期开发弥补。

但是在面临抉择的时候，就需要用到经济学的思维了。为了更好地权衡投入和产出的关系，需要考虑学习成本、文档的全面程度、是否易用、是否稳定等因素。这正是最后抛弃 Dojo，转而投入 jQuery 怀抱的主要原因。jQuery 上手简单，并且目前的核心已经稳定，效率也是所有类库中最优秀的。因为使用广泛，有丰富的学习资源，此外 jQuery 也是插件最多的脚本类库，它丰富的插件资源让我们总可以找到自己想要的功能插件而无须重新开发。所以综合考虑，jQuery 仍是最佳选择！

## 1.2 上手 jQuery

工欲善其事，必先利其器。在开始 jQuery 实例前，先要做了解 jQuery 的各种版本区别、准备好 jQuery 类库文件、搭建 jQuery 的开发环境。

### 1.2.1 jQuery 版本介绍

jQuery 的官方网站是 [www.jquery.com](http://www.jquery.com)，下载地址为：[http://docs.jquery.com/Downloading\\_jQuery](http://docs.jquery.com/Downloading_jQuery)。

jQuery 有不同的版本，比如 1.2.6、1.3.2、1.4 等，本书以 1.4.2 版本为例，在网站上可以下载到如表 1-2 所示的文件。



表 1-2 jQuery 文件列表

版本类型	文件名称	说 明
Uncompressed	jquery-1.4.2.js	未压缩版本的类库文件
Minified	jquery-1.4.2.min.js	压缩后的类库文件
Visual Studio	jquery-1.4.2-vsdoc.js (修改自jquery-1.4.1-vsdoc.js)	带有完整注释的类库版本, 在 Visual Studio 中使用 jQuery 方法时会提示更加全面注释信息。目前 1.4.2 版本没有提供此文件, 可以使用 1.4.1 版本的此文件

在 Visual Studio 2010 中, 对 jQuery 的支持堪称完美。在网站上一一般会引用最小的 Minified 版本的 jQuery 类库文件, 以加快用户的下载速度。但是在程序开发中, 有希望使用 Visual Studio 版本的类库文件, 因为它提供了全面的智能提示。在 Visual Studio 2010 中, 将 Minified 版本和 Visual Studio 版本的 jQuery 类库文件放在同一个目录下, 比如放在同一个 js 目录下:

```
js\jquery-1.4.2.min.js\jquery-1.4.2-vsdoc.js
```

只需要在页面上, 添加对 “jquery-1.4.2.min.js” 文件的引用, Visual Studio 2010 会自动加载 “jquery-1.4.2-vsdoc.js” 文件, 并在使用 jQuery 的函数时提供完整的智能提示, 如图 1-4 所示。

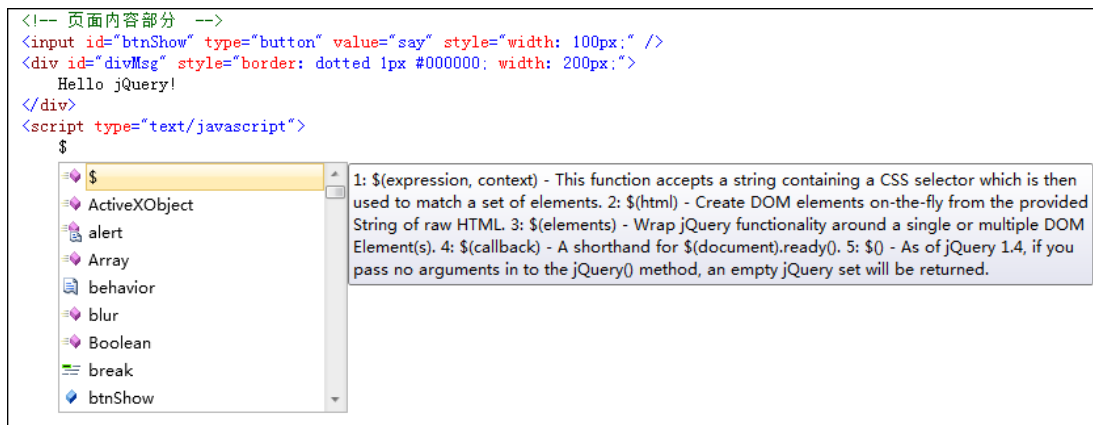


图 1-4 jQuery 智能提示效果

虽然使用 Visual Studio 2010 和 jQuery 1.4 的配合十分完美, 但是在 1.3.2 版本的 jQuery 及更早版本的 Visual Studio 中, 支持得就不尽如人意。比如某些版本有时无法识别 Visual Studio 版本的 jQuery 类库, 或者引用 Visual Studio 版本的类库出现脚本错误 (改成 Minified 版本就正常) 等。所以推荐开发人员使用最新的 jQuery 类库。因为 jQuery 1.4 版本及 Visual Studio 2010 的兼容性都做得很好, 不会给升级带来太多的麻烦。

## 1.2.2 在 Visual Studio 中使用 jQuery

本书的大部分例子都是用 Visual Studio 2010 作为开发环境。Visual Studio 是目前最优秀的 IDE (继承开发环境) 之一, 主要用于 .NET 程序的开发。

首先需要说明, Visual Studio 2005 及其以前的版本, 不支持 jQuery 的智能提示。Visual Studio 2008 和 Visual Studio 2010 版本支持 jQuery 的智能提示功能。其实这源自于这两个



版本的 Visual Studio 对于所有 js 脚本的提示识别的提升，而并非仅仅针对 jQuery。

1.2.1 节已经介绍了在 Visual Studio 2010 中如何启用脚本的智能提示。只需要将 Minified 版本和 Visual Studio 版本的 js 文件放在同一个目录下并引用 Minified 版本的 js 文件即可。

但是在 Visual Studio 2008 中，可能会遇到无法启用 jQuery 智能提示的情况，通常是因为没有为 Visual Studio 2008 打上最新的补丁导致的。如果发现此问题，可以使用下面的方法为 Visual Studio 2008 启用脚本智能提示。

(1) 要确定安装的是 Visual Studio 2008，并且安装了 sp1 补丁。

Visual Studio 2008 sp1 补丁可以在下列地址获取到：<http://msdn.microsoft.com/en-us/vstudio/cc533448.aspx>。

(2) 安装 VS 2008 Patch KB958502 以支持“-vsdoc.js”Intellisense 文件（非必需）。

如果安装了最新版本的 sp1 后仍然无法出现智能提示，则可以尝试安装此补丁。该补丁会导致 Visual Studio 在一个 JavaScript 库被引用时，查找是否存在一个可选的“-vsdoc.js”文件，如果存在的话，就用它来驱动 JavaScript 智能提示引擎。这些加了注释的“-vsdoc.js”文件可以包含对 JavaScript 方法提供了帮助文档的 XML 注释，以及对无法自动推断出的动态 JavaScript 签名的另外的代码智能提示。下载地址为：<http://code.msdn.microsoft.com/KB958502/Release/ProjectReleases.aspx?ReleaseId=1736>。

通过上面的两步，就可以为 Visual Studio 2008 启动 jQuery 智能提示。使用智能提示将极大地加快开发效率。

### 1.2.3 在 Aptana 中使用 jQuery

Aptana Studio 是一个基于 Eclipse 的集成式 Web 开发环境，其最广为人知的就是它非常强悍的 JavaScript 编辑器和调试器。而 jQuery 是一款非常流行的 JavaScript 库，它极大地简化了 JavaScript 的开发。两者的强强联合必能提高 Web 开发人员的工作效率。

可以在 Aptana 的官方网站上下载到其最新版本：<http://www.aptna.com/>。

下面以 Aptana 1.5.1 版本为例来讲解配置的过程。

(1) 启动后首先执行 window|Show Aptana View| Plugins Manager 操作，打开“Plugins Manage”对话框，如图 1-5 所示。

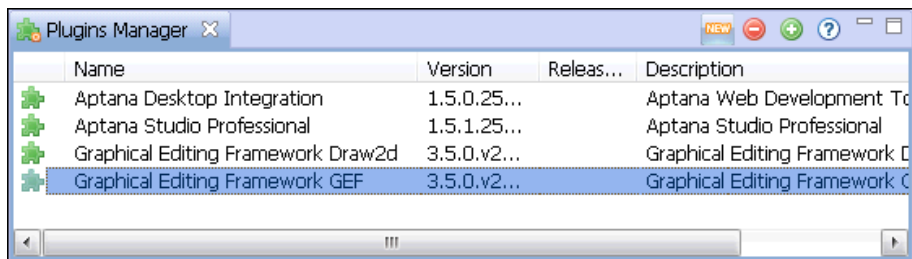


图 1-5 Plugins Manager 对话框

(2) 单击绿色的 Install new feature 按钮，在弹出的 Install Additional Features 对话框中，选中 Ajax Libraries 分类中的 jQuery Support，如图 1-6 所示。

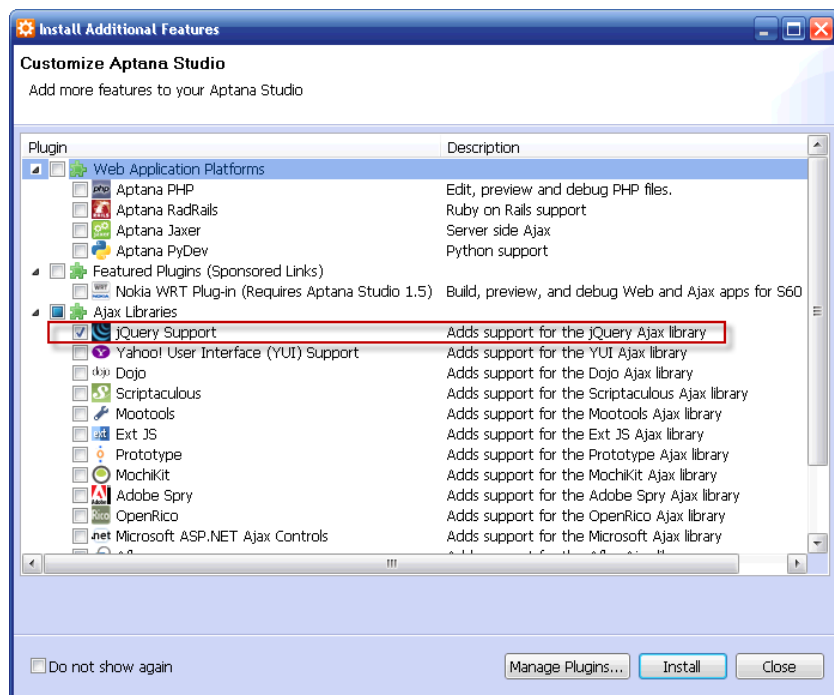


图 1-6 Install Additional Features 对话框

(3) 单击 Install 按钮后在弹出的对话框中，选中 Aptana Support for jQuery 复选框。目前 1.5.1 的 Aptana 支持到 1.3.2 版本的 jQuery，如图 1-7 所示。

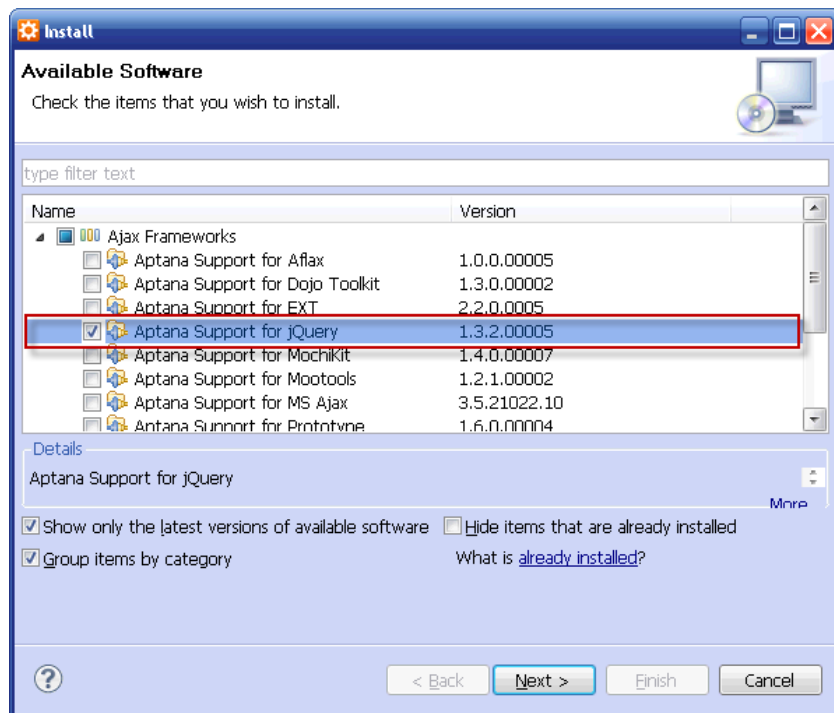


图 1-7 Install 对话框



- (4) 接下来的步骤只需按照提示完成步骤，安装完毕后会要求启动 Aptana。
- (5) 重新启动后，选择 window|preferences 菜单，在弹出的窗口上依次选择：Aptana|editors|javascript|code assist，这样就会发现已经出现了 jQuery1.3，将其勾选，如图 1-8 所示，然后单击 OK 按钮，完成设置。

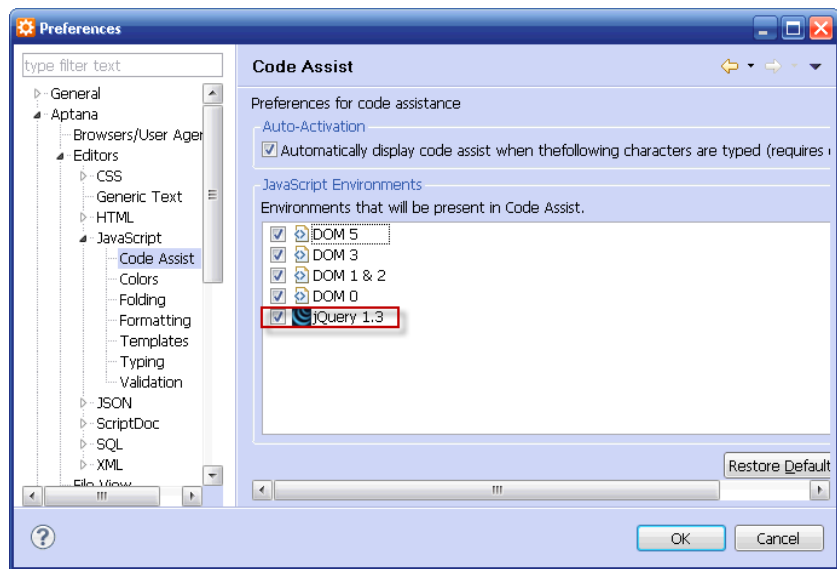


图 1-8 启用 jQuery 智能提示

- (6) 此时可以发现，Aptana 已经为 jQuery 的函数添加了智能提示，如图 1-9 所示。

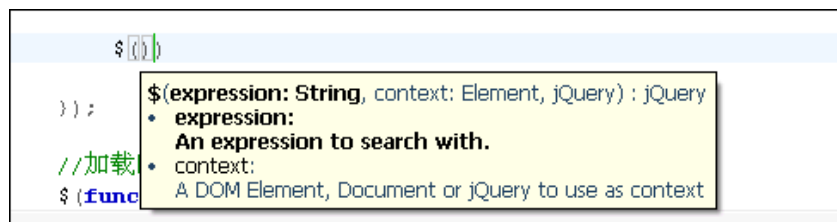


图 1-9 jQuery 智能提示效果

## 1.3 Hello jQuery 实例

本书中的所有实例都使用 ASP.NET 在 Visual Studio 中开发，JavaScript 脚本是与服务器语言无关的，所以使用其他语言的开发人员可以将实例中的服务器语言部分，通常是包含在<%...%>中的部分去掉或者替换成自己熟悉的语言。下面从最简单的 Hello jQuery 开始，来编写第一个 jQuery 的例子。

### 1.3.1 添加脚本引用

首先创建一个名为“Demo-HellojQuery.htm”的页面文件。将 jQuery 的 Minified 版本和 Visual Studio 版本的 js 文件放在同一个目录下，并且在页面上引用 Minified 版本的文件。本





书代码的 jQuery 类库文件，都放在“程序根目录\Static\common\js”文件夹中。所以，需要在 Head 中添加下列引用：

```
<head>
  <title>jQuery - Hello jQuery</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
</head>
```

### 1.3.2 添加 DOM 元素

本实例添加了一个按钮和一个用于显示消息的 Div，代码如下：

```
<input id="btnShow" type="button" value="Say hello to jQuery world" />
<div id="divMsg" style="border:1px dotted #3c3c3c; display:none; width:400px;">Hello jQuery!</div>
```

input 元素是一个按钮，单击时会显示 div 对象。div 对象的默认样式是隐藏的（display 为 none）。

### 1.3.3 创建页面脚本对象

本书的程序都是使用面向对象的方式，组织页面的脚本。在页面的脚本中，创建一个如下结构的对象。

```
<script type="text/javascript">
var thisPage = {
  initialize: function ()
  { //加载时执行
    this.initializeDom();
    this.initializeEvent();
  },
  initializeDom: function ()
  { //初始化 DOM
    this.$btnShow = $("#btnShow");
  },
  initializeEvent: function ()
  { //事件绑定
    this.$btnShow.bind("click", function (event)
    {
      $("#divMsg").toggle(300);
    });
  }
}

$(thisPage.initialize());
</script>
```

其中，thisPage 是一个对象。使用 JSON 格式创建，并为其添加了 initialize、initializeDom、initializeEvent 方法。initialize 是“构造函数”。

initializeDom()方法用于获取页面上的元素，通常使用 jQuery 选择器将页面元素保存在 thisPage 对象的某一个属性中。比如上面的例子，使用 ID 选择器获取了 ID 为 btnShow 的 input 元素，并保存在了 \$bthShow 属性中：

```
this.$btnShow = $("#btnShow");
```

initializeEvent()方法用于为页面元素绑定事件。实例中的语句为“\$bthShow”对象添加



了单击事件，在单击时显示 div 图层：

```
this.$btnShow.bind("click", function (event)
{
    $("#divMsg").toggle(300);
});
```

最后，使用“\$()”方法调用 thisPage 的 initialize()方法，也就是依次执行 initializeDom 和 initializeEvent。“\$()”函数其实是一个事件，使用这个函数调用的方法，会在 DOM 加载完毕、资源文件加载完之前触发。在第 3 章会详细讲解。

现在，单击“Say hello to jQuery world”按钮，会用动画效果显示“Hello jQuery”图层，如图 1-10 所示。

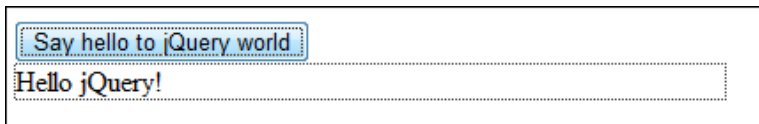


图 1-10 显示 Hello jQuery

在此单击会隐藏此图层。状态和动画的切换主要适用了 jQuery 的 toggle()函数（将在第 9 章“jQuery 动画”详细讲解）。

至此，一个最简单的“Hello jQuery”例子已经开发完毕。

## 1.4 小结

本章介绍了 jQuery 的基础，通过简单的实例更直观地了解了 jQuery 的魅力。也许有经验的开发人员还在为选择哪种脚本框架而犹豫，通过本章脚本类库的对比希望能够解除疑惑。对于网站开发和设计人员而言 jQuery 是最好的选择，其上手容易、文档丰富、功能强大、页面低侵入性、高效率、跨浏览器……有太多的理由让我们使用 jQuery 来改变传统的 JavaScript 开发方式，jQuery 让我们从多浏览器兼容性的噩梦中走出来，从此使开发脚本变得轻松惬意。在后续章节中还会发现 jQuery 更多的魅力，比如使用 jQuery UI 中的组件和皮肤，让一个不懂页面设计的人也可以开发出晶莹剔透的网页和特效。

在全面了解 jQuery 的各类函数之前，第 2 章先学习一些 JavaScript 的精华知识。



## 第 2 章



# 必须知道的 JavaScript 知识

JavaScript 是 jQuery 应用的基础，掌握 JavaScript 这门语言是使用 jQuery 的基础条件。本章不会全面细致地讲解 JavaScript 的全部，而是讲解其精髓，这些知识可以提升大家的 JavaScript 内功。切记要修炼上乘的武功，必须要有深厚的内功基础，否则只可学到其招式而发挥不了功力。JavaScript 实际上包括三部分：

- ❑ ECMAScript 描述了该语言的语法和基本对象。
- ❑ DOM 描述了处理网页内容的方法和接口。
- ❑ BOM 描述了与浏览器进行交互的方法和接口。

本章将讲解 ECMAScript 和 DOM 的相关知识。

## 2.1 JavaScript 基础

通常所说的 JavaScript 语法，实际上是指 JavaScript 中的 ECMAScript 部分。本节主要讲解 JavaScript 的语法和语意特性。

### 2.1.1 JavaScript 与 ECMAScript

很多人知道 JavaScript，却不知道 ECMAScript 为何物。为什么要知道两者的关系呢？因为除了 JavaScript，还有微软的 JScript，以及 Flash 中的 ActionScript，这几种语言在写法上有太多的相似之处。懂得 ECMAScript，就能够清楚地理解这些语言为何如此相似。

什么是 ECMAScript？下面是维基百科中对于 ECMAScript 的定义：

ECMAScript 是一种由 ECMA 国际（前身为欧洲计算机制造商协会）通过 ECMA-262 标准化的脚本程序设计语言。这种语言在万维网上应用广泛，它往往被称为 JavaScript 或 JScript，但实际上后两者是 ECMA-262 标准的实现和扩展。

1995 年 Netscape 公司发布的 Netscape Navigator 2.0 中，发布了与 Sun 公司联合开发的 JavaScript 1.0 并且大获成功。在随后的 3.0 版本中发布了 JavaScript 1.1，恰巧这时微软进军浏览器市场，IE 3.0 搭载了一个 JavaScript 的克隆版——JScript，再加上 Cenvi 的 ScriptEase（也是一种客户端脚本语言），导致了三种不同版本的客户端脚本语言同时存在。为了建立语言的标准，1997 年 JavaScript 1.1 作为草案提交给欧洲计算机制造商协会，第三十九技术委员会（TC39）被委派来“标准化一个通用的，跨平台的，中立于厂商的脚本语言的语法和语意标准”。最后在 Netscape、Sun、微软、Borland 等公司的参与下制订了 ECMA-262，该标准定义了叫做 ECMAScript 的全新脚本语言。

从此以后的 Javascript、JScript 和 ActionScript 等脚本语言都是基于 ECMAScript 标准实现的。

所以，ECMAScript 实际上是一种脚本在语法和语义上的标准。实际上 JavaScript 是由 ECMAScript、DOM 和 BOM 三者组成的。所以说，在 JavaScript、JScript 和 ActionScript 中声明变量，操作数组等语法完全一样，因为它们都是 ECMAScript。但是在操作浏览器对象等方面又有各自独特的方法，这些都是各自语言的扩展。如图 2-1 所示为 ECMAScript 与各个语言的关系。

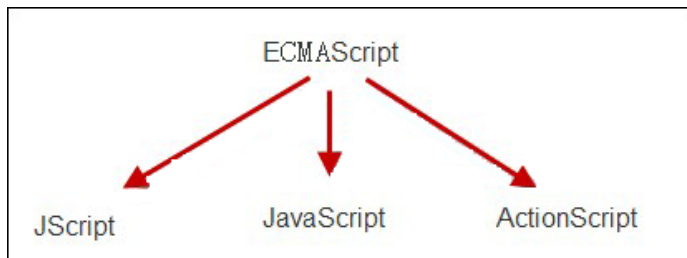


图 2-1 各种脚本语言的关系

### 2.1.2 JavaScript 中的值类型和引用类型

不同于 C#、Java 中多种多样的“类”，JavaScript 是一个相对单纯的世界。JavaScript

中也分为值类型和引用类型两大类，但是这两大类中都只含有很少的几种类型。

都对了解 C#语言的值类型和引用类型不会陌生。JavaScript 中有时也称为原始值（primitive value）和引用值（reference value）。

值类型：存储在栈（stack）中，一个值类型的变量其实是一个内存地址，地址中存储的就是值本身。

引用类型：存储在堆（heap）中，一个引用类型的变量的值是一个指针，指向存储对象的内存处。

### 2.1.3 JavaScript 中的原始类型

时刻记着“值类型”和“引用类型”的区别有助于更好地理解语言的精髓。为了化繁为简，虽然从理论上应该分为“值类型”和“引用类型”，又可以将 JavaScript 中的对象分为“本地对象”、“内置对象”和“宿主对象”，但是在实际应用中为了使 JavaScript 变得真正单纯，可以将 JavaScript 中的类型分为：undefined、null、number、string、boolean、function 和其他 Object 引用类型。

即使是 JavaScript 初学者，对这些类型也不会陌生。这种分类方法前 6 种都是最常使用的 JavaScript 类型，第 7 种 Object 引用类型其实并不是独立的类型，因为 function 就是一种引用类型。另外，JavaScript 中的值类型背后其实也是一个“引用类型”，这一点和 C#极其相似，就是所有的类型都是从 Object 中派生而来。比如 Number 是一个“值类型”，但是其实存在一个引用类型“Number”，可以使用如下方式声明一个变量：

```
var oNumberObject = new Number(55);
```

Number 对象是在 ECMAScript 标准中定义的。但是本文不准备深入地讲解它们，因为最常用的还是使用下面的方式创建一个“值类型”的数值为 55 的变量：

```
var iNumberObject = 55;
```

这就够了不是吗？但是要记住藏在背后的 Number 对象是一个引用类型！

### 2.1.4 undefined、null 和 typeof 运算符

如果对 undefined 和 null 这两种类型经常分辨不清，那么恭喜你，因为你会找到很多的知音。其实要理解这两种类型，首先要知道它们设计的初衷。

undefined：表示一个对象没有被定义或者没有被初始化。

null：表示一个尚未存在的对象的占位符。

有意思的是 undefined 类型是从 null 派生来的。所以它们是相等的：

```
alert(null == undefined); //输出 “true”
```

对于所有的 JavaScript 开发人员，最常碰到的就是对象不存在错误。正如在 C#中的空引用错误一样。很多程序员习惯地以为 JavaScript 中的 if 会自动将 undefined 和 null 对象转化为 false，比如：

```
var oTemp = null;  
if(oTemp){}; //false  
if(undefined){}; //false
```



上面的语句都是正确的，if 中的条件都是 false。但是如果注释掉 oTemp 的声明部分，情况就不同了：

```
//var oTemp = null;
if(oTemp){}; //error
```

程序会抛出错误。但是无论是否声明过 oTemp 对象，使用 typeof 运算符获取到的都是 undefined 并且不会报错：

```
//var oTemp1;
alert(typeof oTemp1); //输出 “undefined”
var oTemp2;
alert(typeof oTemp2); //输出 “undefined”
```

所以，如果在程序中使用一个可能没有定义过的变量，并且没有使用 typeof 做判断，那么就会出现脚本错误。而如果此变量是 null 或者没有初始化的 undefined 对象，可以通过 if 或者 “==” 来判断。切记，未声明的对象只能使用 typeof 运算符来判断！

正因为如此，typeof 经常和 undefined 变量一起使用。typeof 运算符返回的都是一个字符串，而程序员经常会当做类型来使用。是否你也犯过如下错误呢？

```
//var oTemp;
if(typeof oTemp == undefined){...}; //false
```

这里 if 将永远是 false。要时刻铭记 typeof 返回的是字符串，应该使用字符串比较：

```
//var oTemp;
if(typeof oTemp ==undefined){...}; //true
```

下面是 typeof 运算符对各类型的返回结果。

- ☐ undefined: undefined。
- ☐ null: object。
- ☐ string: string。
- ☐ number: number。
- ☐ boolean: Boolean。
- ☐ function: function。
- ☐ object: object。

结果只有 null 类型让人吃惊。null 类型返回 object，这其实是 JavaScript 最初实现的一个错误，然后被 ECMAScript 沿用了，也就成了现在的标准。所以需要将 null 类型理解为“对象的占位符”，就可以解释这一矛盾了，虽然这只是一“种”“辩解”。对于代码编写者一定要时刻警惕这个“语言特性”，因为：

```
alert(typeof null == “null”); //output false
```

永远为 false。

还要提醒，一个没有返回值的 function(或者直接 return 返回)实际上返回的是 undefined。

```
function voidMethod()
{
    return;
}
alert(voidMethod()); //输出“undefined”
```

### 2.1.5 变量声明

因为 JavaScript 是弱类型语言，所以在变量的声明上体现了与 C#等强类型语言的明显不同。

JavaScript 可以使用 `var` 显式地声明变量：

```
var iNum;  
var sName;
```

也可以在一个 `var` 语句中声明多个变量，用“,”分割变量：

```
var iNum, sName;
```

变量的类型是在赋值语句中确定的，JavaScript 使用“=”赋值，可以在声明变量的同时对其进行赋值：

```
var sName="ziqu.zhang";
```

因为是弱类型语言，即使变量的类型在初始化时已经被确定，但仍然可以在之后把它设置成其他类型，比如：

```
var sName="ziqu.zhang";  
alert(sName); //输出“ziqu.zhang”  
sName = 55;  
alert(sName); //输出“55”
```

变量除了可以显式声明，也可以隐式声明。所谓隐式声明，就是不使用 `var` 关键词声明，而直接为变量赋值，比如：

```
//var sName;  
sName="ziqu.zhang"  
alert(sName); //输出“ziqu.zhang”
```

上面的语句不会出任何的错误。就如同使用 `var` 声明过变量一样。但是不同之处是变量的作用域。隐式声明的变量总是被创建为全局变量。即使是在一个函数中创建的变量也依然是全局的。比如：

```
function test()  
{  
    sName = " ziqu.zhang ";  
}  
//var sName;  
test();  
alert(sName); //输出“ziqu.zhang”
```

虽然 `sName` 是在函数中创建的，但是在函数外层仍然能够访问 `sName`，因为 `sName` 是全局变量。

变量可以隐式声明，但是不可以不声明。如果一个变量既没有隐式声明，也没有显式声明，那么在使用时会发生对象未定义的错误。

### 2.1.6 JavaScript 命名规范

因为 JavaScript 语言的灵活性，很多人会忽视变量的命名，有的公司虽制定了后段代码如 C#的命名规范，但却忽视了 JavaScript 变量的命名规范。





从一开始就制定完整的 JavaScript 命名规范是很有必要的，尤其是在越来越追求用户体验的今天，JavaScript 将会承载越来越多的用户逻辑。

先来学习三种命名方法。

- ❑ **Camel 命名法**：首字母小写，接下来的每个单词首字母大写。比如 `var FirstName, var MyColor`。
- ❑ **Pascal 命名法**：首字母大写，接下来的每个单词首字母大写。比如 `var FirstName, var MyColor`。
- ❑ **匈牙利类型命名法**：在以 Pascal 命名法的变量前附加一个小写字母来说明该变量的类型。例如 `s` 表示字符串，则声明一个字符串类型的变量为 `var sFirstName`。

在 JavaScript 中应该使用匈牙利命名法命名变量，使用 Camel 命名法命名函数。

这一点是和 C#、Java 不同的。通常服务器端语言都是用 Pascal 命名法命名，即首字母要大些。但是 JavaScript 中的所有方法首字母都是小写的，为了和 JavaScript 中的默认命名一致，所以要采用 Camel 命名法命名。比如：

```
function testMethod(){}
```

虽然 JavaScript 中的变量可以变换类型，但是熟悉强类型语言的人都知道，这是很危险的做法，很有可能最后在使用时都无法确定变量的类型。所以应该尽量使用匈牙利命名法命名变量，下面是匈牙利命名法的前缀列表，如表 2-1 所示。

表 2-1 JavaScript 中的匈牙利命名法前缀

类 型	前 缀	示 例
Array	a	aNameList
Boolean	b	bVisible
Float	f	fMoney
Function	fn	fnMethod
Int	i	iAge
Object	o	oType
Regexp(正则表达式)	re	rePattern
string	s	sName
可变类型	v	vObj

使用匈牙利命名法是推荐的做法。但是很多程序因为历史遗留原因都使用 Camel 命名法。记住匈牙利命名法才是正确的方法。写代码也是一门艺术，只要有机会就应该做正确的事。

### 2.1.7 变量的作用域与闭包

变量的作用域就是变量作用的范围，只有在变量的作用域内才可以访问该变量，否则是无法访问此变量的。比如：

```
function test()
{
    var sName = " ziqu.zhang ";
}
alert(sName); //输出"sName 未定义";
```



会提示错误"sName"未定义。

JavaScript 变量的作用域基本上与 C#、Java 等语言相同。但是因为 JavaScript 语言的特殊性，有些需要特殊理解的地方。

首先要了解，全局变量是 Window 对象的属性。

前面已经提到过隐式声明的变量都是全局变量。说是“全局”但实际上还是有作用域的，那就是当前窗口 Window 对象。一个全局变量就是 Window 对象的一个属性：

```
function test()
{
    sName = " ziqiu.zhang ";
}
//var sName;
test();
alert(sName);
alert(window.sName);
```

隐式声明了一个全局变量 sName 后，既可以直接访问 sName，也可以通过 window.sName 访问，两者的效果是相同的。

如果说全局变量还是传统的作用域模型，那么闭包（closure）的概念会让初学者迷惑。闭包的概念比较难理解，先看其定义：

闭包是一个拥有许多变量和绑定了这些变量的环境的表达式（通常是一个函数），因而这些变量也是该表达式的一部分。

简单表述：

闭包就是 function 实例以及执行 function 实例时来自环境的变量。

无论是定义还是简单表述，都让人难以理解。因为这都是抽象理论的。通过实例就可以快速地理解闭包的含义：

```
【代码路径：jQueryStorm.Web/chapter2/closure.aspx】
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title> jQuery storm - 闭包举例</title>
</head>
<body>
    <div id="divResult">
    </div>
    <script type="text/JavaScript">
        function start()
        {
            var count = 10;
            //设置定时器，每隔 3 秒钟执行一次
            window.setInterval(function()
            {
                //设置定时器，每隔 3 秒钟执行一次
                document.getElementById("divResult").innerHTML += count + "<br/>";
                count++;
            }, 3000);
        };
        start();
    </script>
</body>
```

上面的实例使用 setInterval() 函数设置了一个定时器，每 3 秒钟会向 divResult 容器中添





加 count 变量的值，并且 count 变量自增。

count 是 start()函数体内的变量，按照通常的理解为 count 的作用域是在 start()函数内，在调用 start()函数结束后应该也会消失。但是此示例的结果是 count 变量会一直存在，并且每次被加 1，数据结果是：

```
10
11
12
13
```

这是因为 count 变量是 setInterval 中创建的匿名函数（也就是包含 count++的函数）的闭包的一部分！

再通俗地讲，闭包首先就是函数本身，比如上面这个匿名函数本身，同时加上在这个函数运行时需要用到的 count 变量。

JavaScript 中的闭包是隐式创建的，不像其他支持闭包的语言那样需要显式创建。在 C# 语言中很少碰到，是因为 C#中无法在方法中再次声明方法。而在一个方法中调用另一个方法通常使用参数传递数据。

JavaScript 中的闭包是非常强大的，可以用于执行复杂的逻辑。但是在使用时要时刻小心，因为闭包的强大也导致了它的复杂，使用闭包会让程序难以理解和维护。

## 2.2 悟透 JavaScript 中的 function

function 类型是 JavaScript 的灵魂，因为程序是由数据和逻辑两部分组成的，基本的数据类型可以存储各种数据，而 function 则用来存储逻辑。当然这只是将 function 当做“方法”来看待而已。实际上 function 类型有着更加强大的生命力。

### 2.2.1 使用 function 声明方法和类型

可以使用 function 声明一个方法，比如：

```
function testMethod()
{
    alert("Hello world!");
}
testMethod(); //输出 “Hello world!”
```

调用该方法将显示“Hello world!”。然而除了方法，function 还可以用来声明“类型”。JavaScript 中本没有“类型”的概念，也就是 C#中的 Class 的概念，但是可以使用 function 伪装一个类型。比如：

```
function Car()
{
    this.color = "none";
    if (typeof Car._initialized == "undefined")
    {
        Car.prototype.showColor = function()
        {
            alert(this.color);
        }
    }
}
```



```
    }  
    Car_initialized = true;  
};
```

上面的代码声明了一个 Car 类，并且带有一个 color 属性和一个 showColor()方法。这里使用的是“动态原型”方法，此方法将在 2.3 节中做详细讲解。

好了，现在可以使用“new”运算符创建一个类的实例，并使用它：

```
var car = new Car();  
car.showColor(); //输出 “none”  
car.color = "blue";  
car.showColor(); //输出 “blue”
```

## 2.2.2 function 的本质

虽然 function 有这么多的能力，但是其本质却很简单。function 变量是引用类型，内容存储的是指向 function 内容的指针，function 的内容就是函数体。

JavaScript 中的对象有一个特点，里面存储的都是 name/value（名/值）。用 name/value 存储属性比较容易理解，一个属性的名称就是 name，比如 color 属性。属性的值就是 value，比如 color 的属性值 red。这与 C#语言是一样的。但是 JavaScript 中的 function 也是一个 name/value，这一点十分独特。可以使用 alert()方法输出 function 变量的内容：

```
alert(testMethod);
```

输出结果如图 2-2 所示。



图 2-2 函数变量输出结果

发现 alert 出来的内容就是 testMethod 的方法体本身。如同一个属性 alert 出来的是其属性值。认清这一点有助于更好地理解 function 的本质。

## 2.2.3 new 运算符

function 类型的对象配合 new 运算法，可以创建一个实例。依然是上面创建一个 Car 类实例的例子：

```
var car = new Car();
```

首字母大写的 Car 是一个函数，但是使用 new 运算符生成的是一个 object：

```
alert(typeof car); //output object
```





而 Car 函数本身则更像是一个类的构造函数。实际上，new 与运算符的操作等价于下面的语句：

```
var car2 = {}; //建议一个空对象
//将 car2 的原型设置为 Car.prototype,这一步是通过 JavaScript 内部的 Object.create 实现的，但是此函数
使内部函数无法直接访问
Car.call(car2); //修改函数调用的上下文
alert(car2.color);
```

其中第二步无法直接用语句替代，是因为实现这一步是在 JavaScript 引擎内部，方法无法直接调用。

## 2.2.4 function 的 arguments 参数对象

JavaScript 不支持方法的重载，原因是在 JavaScript 中同名的 function 只能有一个，并且 function()函数的参数个数可以是任意的。

虽然无法直接支持“重载”，但是可以通过 arguments 对象伪装重载，让一个 function 根据不同的参数实现不同的功能。

arguments 是在 function 中的特殊对象，通过 arguments 对象可以访问到 function 调用者传递的所有参数信息，比如获取到传入的参数个数：

```
function testMethod()
{
    alert(arguments.length);
}
testMethod(); //output "0"
testMethod("abc"); //output "1"
```

可以通过 arguments 的索引 arguments[index]获取每一个传入的参数值：

```
function myMethod()
{
    alert(arguments instanceof Array);
    if (arguments.length == 0)
    {
        alert("no arguments");
    }
    else if (arguments.length == 1)
    {
        alert("Hello:" + arguments[0].toString())
    }
}
myMethod();
myMethod("ziqu.zhang");
```

显然，上面的方法通过判断参数的个数实现了伪装重载。但是这种实现并不好，因为所有的重载逻辑都集中在一个方法中，并且有令人厌烦的多个 if-else 分支。

但是使用 arguments 可以开发出功能强大、灵活的函数。在开发 jQuery 的插件时就要经常使用 arguments 对象实现函数的伪重载。

## 2.2.5 理解 this 指针

在 C#中，this 变量通常指类的当前实例。JavaScript 则不同，JavaScript 中的“this”是



函数上下文，不是在声明时决定的，而是在调用时决定的。因为全局函数其实就是 `window` 的属性，所以在顶层调用全局函数时的 `this` 是指 `Window` 对象，下面的例子可以很好地说明这一切：

```
<script type="text/JavaScript">
  var oName = { name: "ziqu.zhang" };
  window.name = "I am window";
  //showName 是一个函数，用来显示对象的 name 属性
  function showName()
  {
    alert(this.name);
  }
  oName.show = showName;
  window.show = showName;
  showName(); //输出为 "I am window"
  oName.show(); //输出为 "ziqu.zhang"
  window.show(); //输出为 "I am window"
</script>
```

`showName` 是一个 `function`，使用 `alert` 语句输出 `this.name`。因为 `showName` 是一个全局的函数，所以其实 `showName` 是 `Window` 的一个属性 `window.showName`，调用全局的 `showName` 就是调用 `window.showName`，因为 `Window` 上的 `name` 属性为 “I am window”，所以直接调用 `showName` 和调用 `window.showName` 都会输出 “I am window”。

`oName` 是一个对象，前面已经讲过函数也是一个属性，可以像属性一样赋值，所以将 `showName()` 方法复制给 `oName` 的 `show()` 方法。当调用 `oName.show()` 方法时，也会触发 `alert(this.name)` 语句，但是此时方法的调用者是 `oName`，所以 `this.name` 输出的是 `oName` 的 `name` 属性。

利用 “`this` 指向函数调用者” 的特性，可以实现链式调用。`jQuery` 中大部分都是链式调用。首先实现一个链式调用的例子：

```
var oName = { name: "ziqu.zhang", age: 999 };
oName.showName = function()
{
  alert(this.name);
  return this;
};
oName.showAge = function()
{
  alert(this.age);
  return this;
};
oName.showName().showAge();
```

上面的代码首先输出 “ziqu.zhang”，然后输出 “999”。`oName` 使用链式调用方法分别调用了两个方法，等效于：

```
oName.showName();
oName.showAge();
```

使用链式调用的关键点就是要返回调用者本身，也就是 `this` 指针。

在使用 `this` 时，也常常因为 `this` 指向函数上下文的特性，导致引用的错误，比如：

```
var myClass =
{
  checkName: function() { return true; },
```





```
test: function() {  
    if (this.checkName()) {  
        alert("ZZQ");  
    }  
}  
}  
$("#btnTest").click(myClass.test);
```

上面的例子中试图使用面向对象的方式，将一些方法封装在 `myClass` 中，并为 `btnTest` 对象添加单击事件的事件处理函数。现在如果单击 `btnTest` 调用 `myClass.test()` 方法，会发生错误，提示找不到 `checkName` 方法。`jQuery.proxy` 函数可解决此问题，参见“jQuery 工具函数”一章中的“修改函数上下文”一节的内容。

## 2.3 JavaScript 中的原型

JavaScript 中的原型 (prototype) 是 JavaScript 最特别的地方之一。无论是实现 JavaScript 的面向对象还是继承，使用 prototype 都必不可少。

### 2.3.1 使用原型实现 JavaScript 的面向对象

“原型”表示对象的原始状态，JavaScript 中的每个对象都有一个 `prototype` 属性，但是只有 `Function` 类型的 `prototype` 属性可以使用脚本直接操作。`object` 的 `prototype` 属于内部属性，无法直接操作。`prototype` 属性本身是一个 `object` 类型。一个函数的 `prototype` 上所有定义的属性和方法，都会在其实例对象上存在。所以说 `prototype` 就是 C# 类中的实例方法和实例属性。实例方法和静态方法是不同的，静态方法是指不需要声明类的实例就可以使用的方法，实例方法是指必须要先使用 `new` 关键字声明一个类的实例，然后才可以通过此实例访问的方法。

下面是两种方法的声明方式：

```
function staticClass() { }; //声明一个类  
staticClass.staticMethod = function() { alert("static method") }; //创建一个静态方法  
staticClass.prototype.instanceMethod = function() { "instance method" }; //创建一个实例方法
```

上面首先声明了一个类 `staticClass`，接着为其添加了一个静态方法 `staticMethod()` 和一个动态方法 `instanceMethod`，区别就在于添加动态方法要使用 `prototype` 原型属性。

对于静态方法可以直接调用：

```
staticClass.staticMethod();
```

但是实例方法不能直接调用：

```
staticClass.instanceMethod(); //语句错误，无法运行
```

实例方法需要首先实例化后才能调用：

```
var instance = new staticClass(); //首先实例化  
instance.instanceMethod(); //在实例上可以调用实例方法
```

使用 `prototype` 除了可以声明实例方法，也可以声明实例属性。正因为原型有着如此强



大的功能，所以能够使用原型来实现 JavaScript 的面向对象。目前存在着很多以面向对象的形式创建对象的方法，本书不一一详细讲解，只介绍一种最常用、最容易理解和使用的方法——动态原型方法。

假设要定义一个汽车 Car 类型，具有属性 color 和方法 showColor()，则可以使用下面的方式声明和使用：

```
function Car()
{
    this.color = "none";           //声明属性
                                   //声明方法
    if (typeof Car._initialized == "undefined")
    {
        Car.prototype.showColor = function()
        {
            alert(this.color);
        }
    }
    Car._initialized = true;
};
var car = new Car();
car.showColor(); //输出为 "none"
car.color = "blue";
car.showColor(); //输出为 "blue"
```

动态原型方法的精髓在于使用 prototype 声明实例方法，使用 this 声明实例属性。除了更加接近面向对象的编程方式，这种方式特点就是简单易懂。注意充当“类定义”的 function 并没有带有任何参数，虽然也可以传递参数进去，但是不要这么做。这是为了 2.3.2 节讲解的使用原型链实现继承机制而作的准备。

可以在创立了对象以后再为其属性赋值，这样做虽然麻烦了一点，但是代码简单，易于理解。

### 2.3.2 使用原型链实现继承

除了面向对象的声明方式，在面向对象的世界中最常使用的就是对象继承。在 JavaScript 中可以通过 prototype 实现对象的继承。继续 2.3.1Car 的例子。假设 Car 有两个派生类，一个是 GoodCar，特点是跑得快；一个是 BadCar，特点是跑得慢。但是它们和 Car 一样都具有 color 属性和 showColor 方法。

以 GoodCar 类为例，只要让 GoodCar 的 prototype 属性为 Car 类的一个实例，即可实现类型的继承：

```
GoodCar.prototype = new Car();
```

现在，GoodCar 类已经有了 Car 的所有属性和方法：

```
var goodCar = new GoodCar();
goodCar.showColor(); //输出 "none"
```

实现了继承后，GoodCar 还要实现自己的 run() 方法。同样使用 prototype 实现，下面是 GoodCar 类的完整定义：

```
//GoodCar
function GoodCar()
```





```
{  
  GoodCar.prototype = new Car();  
  GoodCar.prototype.run = function() { alert("run fast"); }  
}
```

需要注意 GoodCar 类自身的方法一定要在实现继承语句之后定义。

为何称这种方式为原型链实现继承呢？因为 GoodCar 的 prototype 是 Car，Car 的 prototype 是 object，也就是说 GoodCar 也具有 object 对象所有的属性和方法。这是一个“链”的结构。

使用原型可以简单快捷地实现 JavaScript 的面向对象和继承。

## 2.4 DOM

DOM 太常用了，以至于很多人操作了 DOM 却不知道使用的是 DOM。从概念上区分 JavaScript 核心语法 DOM 和 BOM 是很有必要的，“学院派”作风能够让 Web 的知识体系结构更加清晰。

### 2.4.1 什么是 DOM

DOM（文档对象模型）是 HTML 和 XML 的应用程序接口（API）。

上面是 DOM 的定义。有人将操作 HTML 理解为操作 DOM，这没有错。但是 HTML 并不是 DOM。操作 HTML 也不是 DOM 的唯一功能。DOM 是一种模型及操作这些模型的 API。DOM 分为不同的部分和级别。按照部分来划分，DOM 包括：

- ❑ Core DOM：定义了一套标准的针对任何结构化文档的对象。
- ❑ XML DOM：定义了一套标准的针对 XML 文档的对象。
- ❑ HTML DOM：定义了一套标准的针对 HTML 文档的对象。

所以 Web 开发人员常说的 DOM 其实是指 HTML DOM。上面列举的其实只是 DOM 的一部分。DOM 标准分为 level 1、2、3，每个标准都规定了不同的功能，并不是所有的浏览器都能实现最高级别的 DOM 标准。Firefox 是目前实现 DOM 标准最优秀的浏览器，但仍在为了实现全部的 DOM 标准而努力。本节不对 DOM 做更细致的讲解，本节的目的是让开发人员正确地理解 DOM 与 HTML DOM 的关系。

### 2.4.2 操作 HTML DOM 对象

通过 JavaScript，可以使用 DOM 操作 HTML 的各个元素。如今的页面标准已经是 XHTML 时代。但是页面的本质依然是 XML，一种特殊的用于页面显示的 XML。无论是 ASP.NET、JavaScript 还是 PHP，任何的服务器页面最后都要输出为 HTML 页面——浏览器只认识 HTML。

一个页面会首先被浏览器解析成 DOM，比如下面就是一个页面的 DOM 结构，如图 2-3 所示。



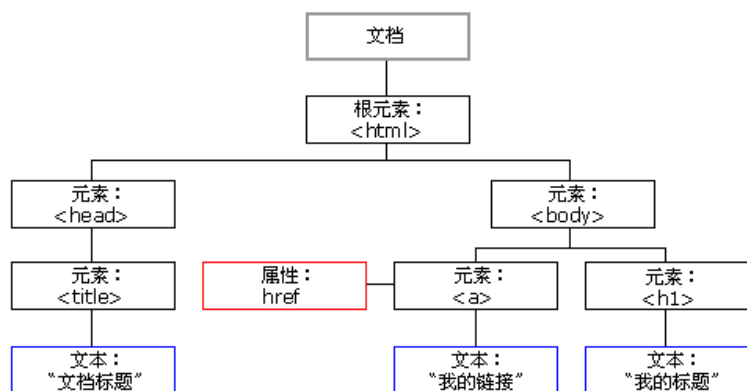


图 2-3 页面的文档对象模型

页面被浏览器解析为 DOM 模型后，通过 JavaScript 操作页面元素，实际上是通过浏览器提供的基于 DOM 的 API 方法实现的。理解了这些就不会提出类似“DOM 元素是 JavaScript 变量吗？”等疑问了。

既然 DOM 是一个文档树，就一定会有根结点：HTML 元素。页面的 document 对象代表文档，documentElement 属性就表示 HTML 对象：

```
var oHtml = document.documentElement;
```

DOM 核心的 API 方法可以获取到一个结点的第一个元素和最后一个元素，因为一个 HTML 对象通常只有 head 和 body 两个元素，所以可以通过下面的方式获取：

```
var oHead = oHtml.firstChild;
var oBody = oHtml.lastChild;
```

理解了树状结构，使用 DOM 核心的方法就可以获取到页面上任意的一个元素。下面是最常用的获取元素结点的方法：

### (1) getElementById

根据元素 ID 获取元素。比如获取 ID 为 divMsg 的元素：

```
document.getElementById("divMsg")
```

这是最常使用的获取元素的方法，一个页面首先应该保证每个元素的 ID 是唯一的。因为任何元素都可以添加 ID 属性，所以通过 ID 能够获取到页面上的任何一个元素。这也是效率最高的一个方法。即使使用 jQuery 功能强大的选择器，也应该首选使用 ID 选择器。

但是需要注意 getElementById() 方法只能通过 document 对象调用，使用 oBody.getElementById 会引发错误。

### (2) getElementsByName

通过 name 属性获取一组元素。方法的名字中是复数形式的“Elements”，说明了获取到的是一组元素，通过 Element 的单复数形式区分获取到的是单个对象还是多个对象，是一种便于记忆的方法。getElementsByName 不常使用，因为通常只有表单元素才带有 name 属性。

### (3) getElementsByTagName

通过 HTML 元素的名称获取一组元素。比如页面上所有的 div：

```
var aDiv = document.getElementsByTagName("div");
```





也可以在一个元素上调用 `getElementsByTagName()` 方法，获取到的是这个元素内的元素：

```
var oHtml = document.documentElement;
var oBody = oHtml.lastChild;
var aMyDiv = oBody.getElementsByTagName("div");
```

上面的语句会获得 `body` 元素下所有的 `div` 元素。因为所有 `div` 都是在 `body` 中的，所以其实和 `document.getElementsByTagName("div")` 的结果是相同的。

除了获取，还有设置属性、创建元素、附加元素等各种 DOM 方法。DOM 的系统学习不在本书的范围内，因为随后的章节将介绍使用 jQuery 来控制页面元素，jQuery 的内部也是使用 DOM 实现的，理解了 DOM 的原理有助于更好地学习 jQuery，甚至发现 jQuery 内部未被发现的错误。

### 2.4.3 DOM 元素与 HTML 元素

通过 DOM 接口操作 HTML 元素，实际上并不是直接改变 HTML 代码，中间有浏览器的帮助。在 JavaScript 中操作的都是 DOM 元素，可以使用 JavaScript 动态地创建一个 `div`，然后附加到 `body` 中：

【代码路径：jQueryStorm.Web/chapter2/DOM.aspx】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery Storm - DOM</title>
  <script type="text/JavaScript">
    function onLoad()
    {
      var oHtml = document.documentElement; //获取到 HTML 对象
      var oBody = oHtml.lastChild; //获取到 Body 对象
      var oDiv = document.createElement("div"); //创建一个 div 对象
      var oText = document.createTextNode("Hello DOM"); //创建一个文本结点
      oDiv.appendChild(oText); //将文本结点放到 div 对象中
      oBody.appendChild(oDiv); //将 div 对象（带有文本结点）放到 body 对象中
    }
  </script>
</head>
<body onload="onLoad()">
</body>
</html>
```

上面的例子使用了标准的 `DOM()` 方法创建了 `div` 元素结点 `oDiv`，以及一个内容结点 `oText`。在 DOM 中内容也是一个结点。

注意，如果要改变页面的 DOM 结构，应该在浏览器加载完页面的 DOM 结构之后进行。否则在 IE 6 中会经常出现浏览器操作终止的错误。本实例将添加元素的工作放在了 `body` 的 `onload` 事件中。使用 jQuery 则会有更好、更简单的处理方式，使用 `$(function(){...})` 这种形式就可以在 DOM 加载完毕后执行函数内的语句。在“jQuery 核心”一节会详细讲解。

动态地添加了一个 DOM 元素，实际上改变了页面的 DOM 模型，随后浏览器会重新解析 DOM 模型并转换成页面显示，于是页面上出现了“Hello DOM”，就好像一开始就包含在 HTML 代码中一样。

很多时候就是在操作 HTML 元素，为何本节要刻意地强调 DOM 元素与 HTML 元素呢？



这是为了引出关于 DOM 属性与 HTML 属性的区别。顾名思义，DOM 属性就是 DOM 元素的属性，HTML 属性就是 HTML 元素的属性。DOM 是无法直接通过源代码看到的。但是 HTML 元素是实实在在存在于页面的 HTML 代码中的，比如：

```

```

这是一个 HTML 元素，图片的地址就是 img 元素的 src 属性，值为“images/image.1.jpg”。因为 HTML 元素最后都要转化成 DOM 元素供浏览器显示，所以浏览器最后将这部分代码解析成一个 DOM 元素，并且也具有一个 src 属性，只是属性值变成了“http://localhost/images/image.1.jpg”（假设网页在 localhost 站点下）。HTML 中的元素最后会映射成 DOM 元素，而且中间的转化并不是完全一致的，img 元素的图片地址会转化为绝对路径，还有些属性比如“class”转化为 DOM 后会改变属性名称，变成了“className”。

牢记，在 JavaScript 中可以直接获取或设置“DOM 属性”，所以如果要设置元素的 CSS 样式类型，要使用的是 DOM 属性“className”而不是 HTML 元素“class”：

```
document.getElementById("img1").className = "classB";
```

在执行完上面的语句后，ID 为 img1 的元素已经应用了 classB 样式，但是查看网页源代码，发现 HTML 元素的 class 仍然是 classA。

有了 jQuery，就不需要 Web 开发人员刻意地理解 class 和 className 的区别。有关使用 jQuery 操作样式和属性将在后面的章节中详细讲解。

## 2.5 其他 JavaScript 秘籍

要讲解 JavaScript 的全部，可能需要很厚的一本书。既然无法全面讲解，就只能抽取一些“武功秘籍”传授给大家。除了上面讲解的 JavaScript 知识，本节再介绍一些 JavaScript 的关键知识点。

### 2.5.1 数据通信格式 JSON

JSON 是指 JavaScript Object Notation，即 JavaScript 对象表示法。所以说 JSON 其实是一种数据格式，因为 JavaScript 原生的支持所以赋予了 JSON 强大的生命力。比如可以使用下面的语句创建一个对象：

```
var oPerson = {  
    name: "ziqui.zhang",  
    age: 999,  
    school:  
    {  
        college: "BITI",  
        "high school": "No.18"  
    },  
    like: ["mm"]  
};
```

JSON 的语法格式是使用“{”和“}”表示一个对象，使用“属性名称：值”的格式创建属性，多个属性用“，”隔开。





上例中 school 属性又是一个对象。like 属性是一个数组。使用 JSON 格式的字符串创建完对象后, 就可以用 “.” 或者索引的形式访问属性:

```
objectA.school["high school"];
objectA.like[1];
```

JSON 经常在 AJAX 中使用。让服务器端页面只返回 JSON 格式的数据, 使用 JavaScript 的 eval() 方法将 JSON 格式的数据转换成对象, 以便使用 JavaScript 操作。

JavaScript 原生地支持了 JSON 格式, 而各种语言也都有支持 JSON 格式类库。在 .net framework 3.5 中微软已经提供了原生的 JSON 序列化器。JSON 是目前客户端与服务器端交互数据的最好的数据格式。下面提供一个以 .NET Framework 3.5 版本中自带的 JSON 序列化器为基础开发的 JSON 帮助类, 代码也可以在本书光盘的代码中或者网站上找到:

【代码路径: jQueryStorm.Common.Utility\JsonHelper.cs】

```
/*
 * File Name      : JsonHelper.cs
 * Creator       : ziqiu.zhang
 * Create Time    : 2008-10-8
 * Functional Description : JSON 帮助类。
 * 使用限制:
 * (1) 只能在 .NET Framework 3.5 及以上版本使用。
 * (2) 对象类型需要支持序列化, 类上添加 [DataContract], 属性上添加 [DataMember]
 * 使用举例请参见单元测试项目。
 * Copyright (c) eLong Corporation. All rights reserved.
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization.Json;
using System.IO;

namespace jQueryStorm.Common.Utility
{
    /// <summary>
    /// JSON 帮助类。用于将对象转换为 Json 格式的字符串, 或者将 JSON 的字符串转化为对象。
    /// 只能在 .NET Framework 3.5 及以上版本使用。
    /// </summary>
    public class JsonHelper
    {
        /// <summary>
        /// 将对象转化为 JSON 字符串
        /// </summary>
        /// <typeparam name="T">源类型</typeparam>
        /// <param name="instance">源类型实例</param>
        /// <returns>Json 字符串</returns>
        public static string ObjectToJson<T>(T instance)
        {
            string result = string.Empty;
            //创建指定类型的 JSON 序列化器
            DataContractJsonSerializer jsonSerializer = new DataContractJsonSerializer(typeof(T));
            //将对象的序列化为 JSON 格式的 Stream
            MemoryStream stream = new MemoryStream();
            jsonSerializer.WriteObject(stream, instance);
            stream.Position = 0;
            //读取 Stream

```



```

        StreamReader sr = new StreamReader(stream);
        result = sr.ReadToEnd();
        return result;
    }
    /// <summary>
    /// 将 JSON 字符串转化为对象
    /// </summary>
    /// <typeparam name="T">目标类型</typeparam>
    /// <param name="jsonString">JSON 字符串</param>
    /// <returns>目标类型的一个实例</returns>
    public static T JsonToObject<T>(string jsonString)
    {
        T result;
        //创建指定类型的 Json 序列化器
        DataContractJsonSerializer jsonSerializer = new DataContractJsonSerializer(typeof(T));
        //将 JSON 字符串放入 Stream 中
        byte[] jsonBytes = new UTF8Encoding().GetBytes(jsonString);
        MemoryStream stream = new MemoryStream(jsonBytes);
        //使用 JSON 序列化器将 Stream 转化为对象
        result = (T)jsonSerializer.ReadObject(stream);
        return result;
    }
}

```

在使用时，假设有一个 MyClass 的类：

```

public class MyClass
{
    public string MyName
    { get; set; }
    public int Age
    { get; set; }
}

```

使用 JsonHelper 的两个泛型方法 ObjectToJson()和 JsonToObject(), 就可以在对象实例和 JSON 字符串之间序列化和反序列化：

```

//创建一个类实例
MyClass myClass = new MyClass() { MyName = "ziqu.zhang", Age = 99 };
//将对象实例转化为 JSON 字符串
string jsonString = JsonHelper.ObjectToJson < MyClass > (myClass);
//将 JSON 字符串转化为对象实例
myClass = JsonHelper.JsonToObject<MyClass>(jsonString);

```

有关服务器端程序的序列化和反序列化，还有许多种实现方式。JSONHelper 类的实现仅供参考。

### 2.5.2 动态语言——eval

使用 eval()方法可以将 JSON 格式的字符转化为 JavaScript 对象：

```

var sJson = "{ name:'ziqu.zhang' }";
eval(" var oName  =" + sJson);
alert(oName.name); //输出“ziqu.zhang”

```

注意这里的 sJson 对象存储的是 JSON 格式的字符串，这个时候字符串的内容还没有被解析成对象。使用 eval()方法可以将 sJson 字符串转化为对象存储在 oName 对象中。

eval()函数可计算某个字符串，并执行其中的 JavaScript 代码。这使 JavaScript 摇身一变





成了动态语言，可以在运行时构造语句，通过执行 `eval()` 函数，就像上面的解析 JSON 字符一样。

`eval()` 函数是有返回值的：

```
var iNum = eval("5+2");  
alert(iNum); //输出“7”
```

`eval` 强大的功能使 JavaScript 开发人员可以发挥无穷的想象力，实现在一些高级语言中无法实现或者实现起来很困难的功能。要知道在 C# 中使用表达式目录树实现动态功能是多么“深奥”的一件事情。

### 2.5.3 JavaScript 中的逻辑运算符

因为逻辑运算符太常用太普通了，所以很多的程序员会认为无非就是 NOT、AND、OR 这三个运算符，返回布尔值。但是在 JavaScript 中却不仅仅如此，AND 和 OR 还会返回对象。

#### 1. NOT 运算符

NOT 运算符用 “!” 表示，与逻辑 OR 和逻辑 AND 运算符不同的是，逻辑 NOT 运算符返回的一定是 Boolean 值。与很多程序不同之处在于，NOT 运算符不仅仅可以运算 Boolean 类型的对象，任何定义了的对象都可以进行 “!” 运算。这里定义了的对象主要是为了排除“未定义的 `undefined`”对象，因为即使对象的类型是 `undefined`，也有两种情况，一种是未定义，一种是未初始化。未初始化的 `undefined` 类型的对象是可以参与 OR、AND 和 NOT 逻辑运算的，只有未定义的 `undefined` 对象在逻辑运算中会出现脚本错误。这一点目前在一些权威教程的网站上的解释都是有错误的，请读者尤其注意。

NOT 运算符的规则如下：

- ❑ 如果运算数是对象，则返回 `false`。
- ❑ 如果运算数是数字 0，则返回 `true`。
- ❑ 如果运算数是 0 以外的任何数字，则返回 `false`。
- ❑ 如果运算数是 `null`，则返回 `true`。
- ❑ 如果运算数是 `NaN`，则返回 `true`。
- ❑ 如果运算数是未初始化的 `undefined`，则返回 `true`。
- ❑ 如果运算数是未定义的 `undefined`，则发生错误。

NOT 运算符其实和 if 条件语句的行为是一样的，只是结果相反。

#### 2. AND 运算符

AND 运算符用双和号 (&&) 表示。AND 运算符的运算数如果都是 Boolean 类型的对象，那么运算规则就是如果有一个运算对象是 `false`，则返回 `false`。

JavaScript 中的 AND 与 NOT 运算符最特别的地方是运算数不一定是 Boolean 类型，返回的也不一定是 Boolean 值，可能返回对象。

AND 运算符的规则如下：

如果一个运算数是对象，另一个是 Boolean 值，则返回该对象。

- ❑ 如果两个运算数都是对象，则返回第二个对象。





- ❑ 如果某个运算数是 `null`，则返回 `null`。
- ❑ 如果某个运算数是 `NaN`，则返回 `NaN`。
- ❑ 如果某个运算数是未初始化的 `undefined`，则返回 `undefined`。
- ❑ 如果运算数是未定义的 `undefined`，则发生错误。

虽然上面描述了 AND 运算符的行为，但是在应用时常常会碰到下面的疑惑：

```
alert(false && null); //输出 “false”  
alert(true && null); //输出 “null”  
alert(null && false); //输出 “null”  
alert(null && true); //输出 “null”
```

因为有一个对象是“`null`”，所以自然想到要应用规则“如果某个元素是 `null`，返回 `null`”，但是第一行语句却返回 `false`。认为这些规则还具有“优先级”，其实是因为 AND 运算符的最本质逻辑规则：

如果第一个运算数决定了结果，就不再计算第二个运算数。对于逻辑 AND 运算来说，如果第一个运算数是 `false`，那么无论第二个运算数的值是什么，结果都不可能等于 `true`。

也就是说 AND 运算符遇到第一个运算数“`false`”时，就停止计算返回 `false` 了。而如果第一个运算数是“`true`”则还要计算第二个运算数，第二个运算数为“`null`”，所以根据规则返回 `null`。

注意看规则中说明是“返回”的，说明运算到此运算数已经返回结果，不会再继续计算其他运算数。所以当运算数是“`null`”，就已经返回了结果“`null`”，即使第二个运算数是 `false` 也不会参与运算。

### 3. OR 运算符

OR 运算符与 C# 中相同，都由双竖线 (`||`) 表示。如果两个运算数都是 `boolean` 类型，OR 运算符的运算逻辑和在几乎所有的语言中都是相同的，即有一个为“`true`”则返回 `true`。

同样在 JavaScript 中，OR 运算符的运算数不一定是 `Boolean` 类型，返回的也不一定是 `Boolean` 值，可能返回对象。

OR 运算符的规则如下：

- ❑ 如果一个运算数是对象，另一个是 `Boolean` 值，则返回该对象。
- ❑ 如果两个运算数都是对象，则返回第一个对象。
- ❑ 如果某个运算数是 `null`，则返回 `null`。
- ❑ 如果某个运算数是 `NaN`，则返回 `NaN`。
- ❑ 如果某个运算数是为初始化的 `undefined`，则忽略此操作数。
- ❑ 如果某个运算数是未定义的 `undefined`，则发生错误。

当两个操作数都是对象时，OR 和 AND 的规则是不同的，是否感觉记忆困难？其实可以将对象理解为“`true`”，在 AND 中遇到“`true`”时需要继续运算，所以继续计算到第二个操作数。返回的也是第二个操作数即最后一个参与运算的操作数。在 OR 运算时遇到“`true`”即返回，所以返回第一个对象也同样是最后一个操作数。

OR 运算符对于未初始化的 `undefined` 类型的对象处理也比较特别，会忽略此操作数，也可以理解为当做“`false`”处理。如果还有第二个操作数则会继续运算。这一点和 AND 不同，AND 运算中如果碰到了未初始化的 `undefined`，则立刻返回 `undefined`。



再次强调，未定义的 `undefined` 在所有的逻辑运算符中都会出现错误。要尽量避免使用未定义的对象。

理解了 JavaScript 的逻辑运算符，就可以介绍一种在 jQuery 插件开发等地方经常会使用的 JavaScript 技巧。假设有下面的方法：

```
function testMethod(param1)
{
    alert(param1);
    alert(param2); //输出“error”
}
```

`testMethod()` 方法有一个参数 `param1`，因为方法的签名就带有 `param1` 参数，所以 `param1` 和完全未声明过的 `param2` 是不一样的。当调用 `testMethod()` 方法并且不传入参数时，会输出“`undefined`”。即 `param1` 此时在函数体内属于“未初始化的 `undefined`”对象。

如果希望在 `testMethod()` 方法中为 `param1` 参数设置默认值“`abc`”，如果没有传入 `param1` 参数则使用默认值。有两种思路，一是可以通过 `typeof` 或者 `arguments` 对象判断是否传入了 `param1` 参数。

通过 `typeof` 判断：

```
function testMethod1(param1)
{
    if (typeof param1 == "undefined")
    {
        param1 = "abc";
    }
    alert(param1);
}
```

或者通过 `arguments` 对象判断：

```
function testMethod2(param1)
{
    if (arguments.length < 1)
    {
        param1 = "abc";
    }
    alert(param1);
}
```

还可以借助 `OR` 运算符的规则，让语句变得更精简。

```
function testMethod3(param1)
{
    alert(param1 || "abc")
}
```

因为 `param1` 是未初始化的 `undefined`，而字符串“`abc`”是对象，所以会返回“`abc`”对象。而如果传入了 `param1` 则会返回 `param1`。这个 JavaScript 技巧在处理可能为未初始化的 `undefined` 时常常用到。但是在使用可能未定义的 `undefined` 对象时同样会发生语法错误。有些开发人员不通过参数传递数据，而是在方法中通过闭包来使用外层的变量，此时就会有使用未声明的 `undefined` 对象的危险。



## 2.6 小结

本节介绍了 JavaScript 语言的一些精髓，指出了 JavaScript 开发人员的一些常见误区和知识盲点，比如理解 ECMAScript，区分 JavaScript 语言和 DOM，理解 DOM 元素和 HTML 元素的区别等。因为篇幅有限不可能对所有的知识进行讲解。但是本章的学习将为后续学习 jQuery 打下坚实的基础，并且让 Web 开发人员的 JavaScript 知识体系得到升华。



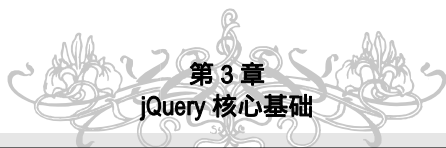


## 第 3 章



# jQuery 核心基础

jQuery 是一套 JavaScript 类库，在开始讲解 jQuery 的各类功能函数前，需要先了解 jQuery 的核心功能和特性，熟悉了 jQuery 的核心特性会让后面的学习更加得心应手。



## 3.1 jQuery 对象

使用 jQuery 的第一件事就是要使用 jQuery 对象,尤其需要注意的是 jQuery 对象和平时 JavaScript 中的 DOM 对象是不同的。初学者常常混淆这两种对象。本节将对 jQuery 对象作深入的解析,以及如何在 jQuery 对象和 DOM 对象之间转化。

### 3.1.1 什么是 jQuery 对象

jQuery 对象在有些书中也被翻译为“jQuery 包装集”,jQuery 将一个 DOM 对象转化为 jQuery 对象后就可以使用 jQuery 类库提供的各种函数。可以将 jQuery 对象理解为一个“类”,并且封装了很多的方法,而且可以动态地通过加载插件扩展这个类,类似于 C# 中的分布类 (partial class)。

除了 jQuery 工具函数, jQuery 的操作都是从 jQuery 对象开始。比如 jQuery 对象上有一个修改元素属性的方法:

```
attr( key, value )
```

要使用此函数,首先要获取 jQuery 对象。假设页面上有一个 img 对象:

```
<img id="myphoto" alt="my photo" src="" />
```

使用下面的语句修改其 src 属性:

```
$("#myphoto").attr("src", "/pic/1.jpg");
```

上面的语句获取了 ID 为“myphoto”的 DOM 元素 (img 元素),并且将其转化为 jQuery 对象,然后使用 attr() 函数将它的 src 属性设置为站点根目录下 pic 文件夹中的“1.jpg”。这个例子中使用了 ID 选择器, jQuery 中的选择器十分强大,第 4 章中将详细讲解。jQuery 选择器获取到的都是 jQuery 对象。

### 3.1.2 jQuery 对象深入解析

jQuery 对象是一个特殊的集合对象。

即使只有一个元素, jQuery 对象仍然是一个集合。说其特殊是因为实际上 jQuery 对象是包含一个集合对象和各种函数的类,注意这里没有称做是“集合 (Array) 的扩展”,是因为 jQuery 并没有扩展 Array 的原型,因为那样会导致所有的 Array 对象都会成为 jQuery 对象。可以使用下面的代码验证这一结论:

```
var array = new Array();  
array.push("123");  
array.push("456");  
alert("Array:" + Object.prototype.toString.call(array)); //output: Array:[object Array]  
alert("jQuery Object:" + $("#myphoto").toString()); //输出 jQuery Object:[object Object]
```

因为 Array 对象重写了 object 的 toString() 方法,所以使用上面的方法可以调用 Array 父类 (object 类) 的 toString() 方法。此技巧可以用来判断一个对象是否为集合。在 jQuery 内部判断是否为集合的 isArray() 方法就是用的这种实现。

图 3-1 是 jQuery 对象的一部分,没有列出来所有的功能函数,但是可以看出来一个



jQuery 对象是由一个存储 DOM 对象的集合、各种功能函数及其他辅助属性组成的。提醒一下这里的集合存储的是 DOM 对象，聪明的读者可能已经想到了如何将 jQuery 对象转化为 DOM 对象了。后面的内容中会详细讲解。



图 3-1 jQuery 对象

jQuery 对象的引用与一般的对象也有区别。一个 jQuery 对象实际上是“jQuery.fn”对象的引用，下面是 jQuery 内部的实现语句：

```
jQuery.fn = jQuery.prototype = { ... }
```

在第 2 章中已经讲解了 JavaScript 中原型的概念。原型相当于“实例方法”，并且原型是一个对象。上面的语句表明“jQuery.fn”是一个 jQuery 对象的引用。然而有趣的是，所有的 jQuery 对象都是 jQuery.fn 对象的实例！下面是获取 jQuery 对象的核心方法：

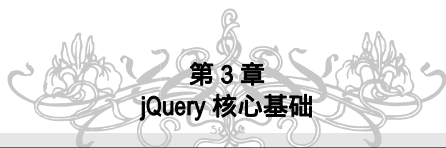
```
jQuery = window.jQuery = window.$ = function( selector, context ) {
    return new jQuery.fn.init( selector, context );
}
```

一个 jQuery 对象实际上就是“jQuery.fn.init()”方法的返回结果。“jQuery.fn.init”返回的是“jQuery.fn”对象。jQuery 的插件开发正是应用这个原理，对“jQuery.fn”对象进行扩展，实现扩展 jQuery 对象的目的。在“插件开发”一章中将详细讲解。

### 3.1.3 jQuery 对象转换为 DOM 对象

只有 jQuery 对象才能调用 jQuery 类库的各种函数，同样有些 DOM 对象的属性和方法在 jQuery 上也是无法调用的，不过基本上 jQuery 类库提供的函数包含了所有的 DOM 操作。有时尤其是在初学 jQuery，无法记住 jQuery 的所有函数时，会有很长一段时间使用 jQuery 选择器配合原始的 DOM 函数进行开发。所以两种对象的转化是很有必要的。

在 3.1.2 节对 jQuery 对象的深入解析时就已经发现，jQuery 对象的索引保存的是 DOM 对象，所以可以通过索引将 jQuery 对象转化为 DOM 对象（实际上是获取保存在 jQuery 对象中的 DOM 对象）。



```
$("#myphoto")[0];
```

通过索引返回 DOM 对象后,就可以使用各种 DOM 对象的方法和属性,比如获取 DOM 对象的 src 属性:

```
alert($("#myphoto")[0].src);
```

如果想要遍历 jQuery 对象中的每个元素,通常使用 each() 函数。

```
each(callback)
```

callback() 是一个回调函数,此函数中的 this 也指向 DOM 元素。

```
$("#myphoto").each(function(i){  
    this.src = "test" + i + ".jpg";  
});
```

对于懒人有一个小窍门,如果读者不想记忆在不同的 jQuery 函数中的 this 到底是 jQuery 对象还是 this 对象,可以使用 3.1.4 节的方法把所有的“this”都转化成 jQuery 对象。因为即使一个对象已经是 jQuery 对象也不会出错。

### 3.1.4 DOM 对象转化为 jQuery 对象

如果已经获得了一个 DOM 对象,可以使用“jQuery( elements )”函数将其转化为 jQuery 对象:

```
var img = document.getElementById("myphoto");  
jQuery(img).css("border", "solid 2px #FF0000");
```

上面的代码中 img 是使用 DOM 获取到的 DOM 对象。将其转化为 jQuery 对象后就可以使用 jQuery 对象的 css() 方法更改其样式。

可以使用“\$”代替“jQuery”,因为在 jQuery 的内部有如下实现:

```
jQuery = window.jQuery = window.$
```

“\$”字符在 JavaScript 中可用做变量名,并且可以作为前缀出现。但是一些其他的类库或者程序可能已经使用了“\$”作为变量名,在后面将介绍 jQuery 如何避免“\$”变量的冲突。在不冲突的情况下,下面的语句是等同的:

```
jQuery(img).css("border", "solid 2px #FF0000");  
$(img).css("border", "solid 2px #FF0000");
```

“jQuery( elements )”函数的 elements 参数还可以是 jQuery 对象,虽然将一个 jQuery 对象再次转化没有意义,这是为了当不确定一个对象的类型是 jQuery 对象还是 DOM 对象时,可以再次调用此函数进行转化,这样可以保证此对象一定是 jQuery 对象。

### 3.1.5 jQuery 对象的链式操作

首先来看一个链式操作的例子:

```
$("#myphoto").css("border", "solid 2px #FF0000").attr("alt", "ziqu is a good man!");
```

对一个 jQuery 对象先调用了 css() 函数修改样式,然后使用 attr() 函数修改属性,这种调用方式像链一样,所以称为“链式操作”。





链式操作能够让代码变得简洁，因为往往可以在一条语句中实现以往多条语句才能完成的任务。比如如果不使用链式操作，需要用两条语句才能完成上面的任务：

```
$("#myphoto").css("border", "solid 2px #FF0000");  
$("#myphoto").attr("alt", "ziqu is a good man!");
```

除了增加了代码量，还调用了两次选择器，降低了速度。

在一个较短的链式操作中，往往语句比较清晰，可以分步骤地对 jQuery 对象实施各种操作。但是一个链式操作不应该太长，否则会造成语句难以理解，因为要查看 jQuery 对象当前的状态并不是容易的事，尤其如果涉及 jQuery 对象中元素的增删操作时更加难以判断。

并不是所有的 jQuery 函数都可以使用链式操作。这与链式操作的原理有关。之所以可以实现链式操作是因为其中的每个函数返回的都是 jQuery 对象本身。在 jQuery 类库的内部实现中，虽然很多的函数都返回 jQuery 对象本身，但都是通过调用内部有限的几个函数实现的，比如 attr() 函数设置属性时，实际上最后调用了 "jQuery.each(object, callback, args)" 方法。注意此方法不是 jQuery 对象方法，jQuery 对象方法也有一个 each() 函数，为 "jQuery.fn.each(callback, args)"，此函数最后同样调用 jQuery.each 函数：

```
each: function( callback, args ) {  
    return jQuery.each( this, callback, args );  
},
```

下面看一看 jQuery.each 函数的返回结果：

```
each: function( object, callback, args ) {  
    //省略  
    return object;  
}
```

object 是 jQuery.fn 对象，即 jQuery 对象。最后返回的还是 jQuery 对象。

可以使用下面的原则判断一个函数返回的时候是 jQuery 对象，即是否可以用于链式操作。

除了获取某些数据的函数，比如获取属性值 "attr(name)"，获取集合大小 "size()" 这些函数明显是返回数据的。除了这些函数之外的 jQuery 函数都可以用于链式操作，比如设置属性 "attr(name,value)"。

### 3.1.6 “\$”变量的使用

“\$”变量是“jQuery”变量的引用。“jQuery”变量是全局变量，jQuery 对象是指“jQuery.fn”，不要混淆。“jQuery”变量类似于静态类，上面的方法都是静态方法，可以在任何时刻调用。比如 “jQuery.each”。“jQuery.fn”是实例方法，只能在 jQuery 对象上调用。比如 "jQuery.fn.each()" 方法只能通过 "\$('#id').each()" 这种形式调用。

前面已经提到，可以使用 “\$” 代替 “jQuery”，因为在 jQuery 的内部有如下实现：

```
jQuery = window.jQuery = window.$
```

所以 “\$” 变量和 “jQuery” 变量实际上是 Window 对象的属性，也就是全局变量。可以在页面上的任何地方调用。



## 3.1.7 解决多类库冲突——“\$”变量冲突问题

因为在其他的 JavaScript 类库中，也会使用“\$”变量作为类库对象的引用，比如 Prototype。当一个页面需要同时使用两个脚本库时，就会产生冲突，导致“\$”变量的引用不明确。

jQuery 提供了 jQuery.noConflict() 及其重载用来解决此问题，jQuery.noConflict 函数有两个重载：

jQuery.noConflict() 和 jQuery.noConflict( extreme )，虽然官方类库提供了两个重载方法，但是 JavaScript 并不支持重载，内部实现仅仅是一个函数，根据是否传入了参数而执行不同的处理。

jQuery.noConflict() 的作用是将变量 \$ 的控制权转交给第一个实现它的那个库。如果 extreme 参数为“true”即表示同时将“jQuery”变量的控制权也转交出去。也许有的脚本中不仅仅占用了“\$”变量而且连“jQuery”变量也占用了，比如：

```
jQuery.noConflict(true);
```

下面是一个同时使用 jQuery 类库和 Prototype 类库的例子：

【代码路径：jQueryStorm.Web/chapter3/noConflict.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery Storm - noConflict 函数实例</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
  <script src="../../Static/common/js/prototype.js" type="text/javascript"></script> </head>
<body>
  <!-- 页面内容部分 -->
  <div id="divMsg" style="border:solid 1px #000000; padding:20px;">
    [ 输出控制台 ]<br />
  </div>
  <!-- 尾部脚本块 -->
  <script type="text/javascript">
    jQuery.noConflict();
    $("#divMsg").innerHTML = "written by Prototype"; //调用 Prototype
  </script>
</body>
</html>
```

在上面的例子中先引用了 Prototype 的类库，再引用了 jQuery 类库。因为 JavaScript 文件是顺序加载的，所以如果不使用 jQuery.noConflict()，“\$”变量将被 jQuery 使用。使用“\$”调用 Prototype 类库时将失效。使用 noConflict 函数后，“\$”变量重新被 Prototype 类库使用。

jQuery.noConflict() 函数返回原“jQuery”变量本身，应用此原理可以改变“jQuery”变量的引用名称：

```
$$ = jQuery.noConflict();
$("#divMsg").innerHTML = "written by Prototype"; //调用 Prototype
$("#divMsg").html( $("#divMsg").html() + "<br/>" + "written by jquery"); //调用 jQuery
```

上例中将“jQuery”变量的引用赋给了“\$\$”变量，则可以使用“\$”调用 Prototype 类库，使用“\$\$”调用 jQuery 类库。

通过上面的原理解讲，很容易地想到另外两种解决冲突的方法：





1. 在“jQuery”变量不冲突的前提下，页面中不使用“\$”变量，全部用“jQuery”变量代替。

2. 先引入 jQuery 类库，再引入其他类库。此时“\$”变量的控制权会被其他类库占用。如果使用这种方式则千万不能再调用 jQuery noConflict() 函数。

如果一定想要使用“\$”变量引用 jQuery 类库，在插件开发中常常使用下面的技巧：使“\$”变量作为参数，只在函数内部代表 jQuery 的引用，不影响全局的“\$”变量：

```
(function($){
{
    $("#divMsg").html($("#divMsg").html() + "<br/>" + "written by jquery"); //调用 jQuery
})(jQuery);
```

上面的语法其实是首先声明了一个匿名函数，接受一个参数。紧接着立刻调用了此函数并且将“jQuery”变量作为参数传入，则函数中的“\$”就代表了“jQuery”变量。注意 function 前后的“(”和“) ”不能省略，否则将不能执行函数。

## 3.2 jQuery 文档处理程序

所谓文档处理程序，是指 jQuery 中提供的 \$(document).ready 事件。此事件会在 DOM 加载完毕后触发，而 window.onload 和 body.onload 都是在页面完全加载后触发。本节将对 jQuery 文档处理程序做详细讲解。

### 3.2.1 jQuery 文档处理程序介绍

在 jQuery 中想实现在 DOM 加载完毕后的处理逻辑，只需要使用 \$(document).ready() 事件。此事件函数称为“jQuery 文档处理程序”。

可以在页面的任何位置，甚至是外部的 js 文件中，编写下列语句：

```
$(document).ready(function() { alert("document.ready") });
```

则在页面的 DOM 加载完毕后，会立刻执行 alert 语句。如果页面上有大的图片、js 文件等外部资源需要加载，jQuery 的文档处理程序会在其之前执行，而 window.onload 和 body.onload 是在所有的资源文件加载完毕后执行的。

ready() 函数是 jQuery “事件函数”中提供的一个 jQuery 对象函数，签名为：ready( fn )。fn 是 ready 事件发生时执行的函数。

因为是 jQuery 对象函数，意味着可以在任何 jQuery 对象上调用：

```
$("body").ready(function() { alert("body.ready") });
```

上面的语句等同于：

```
$(document).ready(function() { alert ("body.ready") });
```

虽然 ready() 函数可以作用在任何 jQuery 对象上，但是使用时一定要注意对象是否具有 ready 事件。

也可以使用“\$(fn)”这种简化的形式：

```
$( function() { ... } ) //等效于 $(document).ready(function() { ... });
```



同 jQuery 中所有的事件对象一样, `$(document).ready` 事件是多播事件, 会按照出现的先后顺序执行。

比如, 可以调用两次 `$(document).ready` 事件:

```
$(document).ready(function() { alert("document.ready-1") });  
$(document).ready(function() { alert("document.ready-2") });
```

则在 DOM 加载完毕后, 首先输出 “document.ready-1”, 然后输出 “document.ready-2”。如果使用传统的:

```
window.onload=function(){...};
```

上面的代码会将 `window.onload` 原有的事件处理掉, 然后绑定新的事件。有关事件的更多内容将在 jQuery 事件章节详细讲解。

### 3.2.2 文档处理程序的优势

如果脚本需要在页面加载时执行, 那么大部分的脚本都可以放在 `$(document).ready()` 事件中。

在列举 jQuery 文档处理程序的优势之前, 先看一个常见的 JavaScript 编程错误: DOM 未加载完成即改变 DOM 模型。

在传统的 JavaScript 编程中, 有时会在页面的头部或者底部直接插入 Script 块并编写代码, 比如下面的例子:

```
【代码路径: jQueryStorm.Web/chapter3/WrongWay-DOM.htm】  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
  <title>jQuery Storm - 常见错误编程方式举例</title>  
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>  
</head>  
<body>  
  <!-- 页面内容部分 -->  
  <div id="divMsg" style="border:solid 1px #000000; padding:20px;"></div>  
  <!-- 尾部脚本块 -->  
  <script type="text/javascript">  
    //这条语句在某些情况下会导致错误  
    document.getElementById("divMsg").innerHTML+="<div style='border:solid 2px #FF0000'>动态添加的图层</div>";  
  </script>  
</body>  
</html>
```

此例子在所有的浏览器都运行良好, 但是却存在隐患。因为在页面加载时, 就在 `divMsg` 容器中添加了一个新的 `div` 对象, 也就是添加了一个 DOM 对象。当网速变慢或者页面很大需要一定的加载时间时, 会出现 “中止操作” 的错误, 如图 3-2 所示。





图 3-2 终止操作

在 IE 8 中根据操作会提示类似下面的错误：

HTML Parsing Error: Unable to modify the parent container element before the child element is closed (KB927917)

这个错误甚至曾经在 Google 首页、淘宝等知名网站上都出现过，因为此错误很难在测试时发现。

所以做 Web 开发时要记住一条真理：永远不要在 DOM 加载时修改 DOM 结构。

如果需要在页面加载时修改 DOM 结构，原始的做法是通过 `window.onload` 和 `body.onload` 事件实现，比如针对上面的错误例子，可以修改为：

```
<script type="text/javascript">
window.onload = function()
{
    document.getElementById("divMsg").innerHTML =
    "<div style=\"border:solid 2px #FF0000\">动态添加的图层</div>";
};
</script>
```

具体的实现方式还有很多，应用 `window.onload` 和 `body.onload` 是因为这两个事件都是在 DOM 加载完成并且所有的页面资源加载完成后执行的，这是最简单的做法。或者根据“`document.readyState`”判断 DOM 的状态，如果是“complete”则进行某些操作，jQuery 的文档处理程序 `$(document).ready` 内部正是使用的此原理。

有了 jQuery 文档处理程序，事情变得简单了：

```
$(function(){
    document.getElementById("divMsg").innerHTML = "<div style=\"border:solid 2px #FF0000\">动态
添加的图层</div>";
})
```

这只是使用 jQuery 文档处理程序 `$(document).ready` 的原因之一。下面是所有的优点总结：

- ❑ 避免在 DOM 加载时修改 DOM 结构。
- ❑ 在 DOM 加载后立刻执行，避免了等待页面资源加载的等待时间。
- ❑ `$(document).ready` 为多播事件，不会覆盖其他人绑定的事件。
- ❑ 使用后绑定方式使代码行为和分离。低侵入性。

jQuery 的文档处理程序是 jQuery 的核心，以后应该将页面加载时需要执行的语句都放入 `$(document).ready` 事件中。

### 3.2.3 jQuery 文档处理程序深入解析

jQuery 文档处理程序简化了页面加载时编写 JavaScript 的方式。前面也曾提出，在没有 jQuery 之前其实也存在很多的解决办法，只是 jQuery 使一切变得更加简单易用。本节深入

jQuery 文档处理程序\$(document).ready 的实现方式,也许有一天碰到了不能使用 jQuery 的项目,明白了原理就可以使用原始的 JavaScript 编写自己的轻量级类库的实现。

当然,学会使用 jQuery 文档处理程序也已经足够了,不想深入学习的读者可以跳过本节。

下面是 jQuery 对象上的 ready()函数:

```
ready: function(fn) {
    bindReady();      // 加监听者
    //如果 DOM 已经准备好
    if (jQuery.isReady)
        fn.call( document, jQuery ); // 立刻执行函数
    //否则将函数放到 readyList 中, 等待执行
    else
        jQuery.readyList.push( fn );
    return this;
}
```

分析此函数,发现可以在页面加载完成后调用 ready 事件,此时传递的函数会立刻执行。如果页面还没有加载完毕,则将传入的方法放入 jQuery.readyList 集合中,等待 DOM 加载完毕后一起调用。具体的调用逻辑是放在 bindReady()函数中的,下面是核心的 bindReady 函数实现:

```
var readyBound = false;
function bindReady(){
    if ( readyBound ) return;
    readyBound = true;

    //Mozilla, Opera 和 webkit 支持
    if ( document.addEventListener ) {
        //为 DOMContentLoaded 事件添加事件处理函数
        document.addEventListener( "DOMContentLoaded", function(){
            document.removeEventListener( "DOMContentLoaded", arguments.callee, false );
            jQuery.ready();
        }, false );
    }
    //处理 IE 事件模型
    } else if ( document.attachEvent ) {
        document.attachEvent("onreadystatechange", function(){
            if ( document.readyState === "complete" ) {
                document.detachEvent( "onreadystatechange", arguments.callee );
                jQuery.ready();
            }
        });
        // 如果是 IE 并且不是 iframe, 则继续检查
        if ( document.documentElement.doScroll && window == window.top ) (function(){
            if ( jQuery.isReady ) return;
            try {
                document.documentElement.doScroll("left");
            } catch( error ) {
                setTimeout( arguments.callee, 0 );
                return;
            }
        })();
        jQuery.ready();
    }

    jQuery.event.add( window, "load", jQuery.ready );
}
```



上面是 `bindReady()` 函数的完全代码，因为其中调用了 jQuery 的一些类库函数所以总代码量并不多。`readyBound` 全局变量和 `bindReady()` 全局函数密切相关，`readyBound` 变量的意义是为了保证 `bindReady()` 函数只能被调用一次。

分析代码可以看出，想要捕获到“DOM 加载完毕”这一时间点，在 Firefox、Opera 和 Safari 3.1+ 中提供了“DOMContentLoaded”事件。所以实现起来最简单。

在 IE 中则没有此事件，但是可以通过判断 `document.readyState` 的状态是否为“completed”来判断 DOM 是否加载完毕。通过为 `document` 的 `onreadystatechange` 事件添加处理函数，使 `document` 每次状态改变时都调用处理函数：

```
document.attachEvent("onreadystatechange", function(){
    if ( document.readyState === "complete" ) {
        document.detachEvent( "onreadystatechange", arguments.callee );
        jQuery.ready();
    }
});
```

上面的 `ready()` 函数是 `jQuery.fn.ready`，发现 jQuery 中获取到 DOM 执行完毕的时间点后，都是调用 `jQuery.ready()` 函数，此函数中会调用保存在 `jQuery.readyList` 中的所有方法：

```
jQuery.extend({
    isReady: false,
    readyList: [],
    //当 DOM 加载完毕时的事件处理函数
    ready: function() {
        //确定 DOM 还没有加载
        if ( !jQuery.isReady ) {
            jQuery.isReady = true;                //设置 DOM 状态为已经加载
            //执行函数列表中的所有函数
            if ( jQuery.readyList ) {
                jQuery.each( jQuery.readyList, function(){
                    this.call( document, jQuery );
                });
                jQuery.readyList = null;            //重置函数列表集合
            }
            jQuery(document).triggerHandler("ready");//触发所有绑定的 Ready 事件
        }
    }
});
```

至此 jQuery 文档处理程序的主线已经执行完毕。其中还包含对 IE、iFrame 的特殊处理。可以看出 jQuery 文档处理程序是健壮的，可以兼容多个浏览器。如果开发人员自己处理这些逻辑，很容易因为不知道某些浏览器的特性而产生 bug。此时也许能够理解 jQuery 的口号了：“write less, do more!”

### 3.2.4 jQuery 文档处理程序注意事项

虽然 `$(document).ready` 函数是在 DOM 加载完毕后立刻执行，但是需要注意：

`$(document).ready` 事件中函数语句不能保证一定在资源文件加载前执行。

这和事件中函数的执行时间及资源文件的加载时间都有关系。因为除了 js 文件，其他的资源都是异步加载的，也就是说 `$(document).ready` 中的语句调用和资源文件的加载是同时执行的。

下面是加载程序的例子：

```
【代码路径：jQueryStorm.Web/ chapter3/ ready.htm】
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery Storm - jQuery 文档处理程序</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
  <script type="text/javascript">
    function console(obj)
    {
      document.getElementById("divMsg").innerHTML += obj;
      document.getElementById("divMsg").innerHTML += "<br/>";
    }
  </script>
</head>
<body onload="console('HTML:body.onload')"> //为 body 对象的 onload 事件添加输出
  <!-- 页面内容部分 -->
  <div id="divMsg" style="border:solid 1px #000000; padding:20px;">
    [ 输出控制台 ]<br />
  </div>
  <br />
  //为 img 元素同样添加 onload 输出
  
  
  <!-- 尾部脚本块 -->
  <script type="text/javascript">
    $(document).ready(function() { console("document.ready") });
  </script>
</body>
</html>
```

页面上有两个图片元素，imgSmall 是小图片，加载得很快。imgBig 是大图片，加载得较慢。运行上面的实例，会发现每次刷新的结果是随机的，有多种可能的结果，如图 3-3 和图 3-4 所示的是其中的两种。

```
[ 输出控制台 ]
imgSmall.onload
document.ready
imgBig.onload
HTML:body.onload
```

图 3-3 结果 1

```
[ 输出控制台 ]
document.ready
imgBig.onload
imgSmall.onload
HTML:body.onload
```

图 3-4 结果 2

这两种结果不同之处就在于 document.ready 和两个图片 onLoad 的顺序。浏览器对于图片等静态资源是异步加载的，具体的处理方式上不同的浏览器会有所不同，比如线程数的不同等。因为 imgSmall 是较小的图片，加载速度较快，所以会出现 imgSmall 图片先加载，\$(document).ready 事件后执行的情况，两者的执行顺序是不确定的。图 3-3 就说明了也可能图片文件后加载完成，先执行\$(document).ready 事件。细心的读者会发现图 3-4 中 imgBig 在 imgSmall 之前完成了加载，这是在 IE 6 下，在 FireFox 下则通常是 imgSmall 先加载，因为是异步加载，所以顺序是不确定的。

可以确定的就是 body 的 onload 事件一定是在所有的资源都加载完成后执行的。



所以要时刻铭记，不要使\$(document).ready 事件执行顺序和资源文件的加载产生关系。下面就是一个错误的例子：

```
【代码路径：jQueryStorm.Web/ chapter3/WrongWay-Ready.htm】
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery Storm - 常见错误编程方式举例</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
</head>
<body>
  <!-- 页面内容部分 -->
  <div>
    <a href="http://www.elong.com" target="_blank">超链接，点我！</a>
  </div>
  <!-- 尾部脚本块 -->
  <script type="text/javascript">
    $(function()
    {
      setTimeout(function()
      {
        $("a").click(function(event)
        {
          event.preventDefault(); alert(this.href);
        });
      }, 1000);
    });
  </script>
</body>
</html>
```

在此实例中，有一个<a>元素并且为其设置了 href 链接，默认情况下单击<a>元素会新开一个页面打开此链接，但是页面制作者希望的是在当前页使用弹出框的形式打开此链接。本实例是使用 alert 输出链接地址，在后面的 jQuery UI 章节将详细讲解如何制作真正的弹出层。

实例中使用 setTimeout()函数模拟函数执行缓慢的情况。现在页面存在一个问题：如果在页面加载后 1 秒内单击<a>元素，因为 JavaScript 程序中的语句还没有执行，所以<a>元素的默认行为还没有被改变，仍然会打开一个新窗口。1 秒后单击<a>元素变为弹出提示框。

有下面两种方式可以修正这个错误的应用：

```
【代码路径：jQueryStorm.Web/ chapter3/RightWay-Ready.htm】
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery Storm - 常见错误编程方式举例</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
</head>
<body>
  <!-- 页面内容部分 -->
  <div>
    <a href="http://www.elong.com" onclick="return false;" target="_blank">超链接，点我！</a>
    <br />
    <span rel="href" data="http://www.elong.com" ><u style="cursor:pointer;">超链接，点我！
  </u></span>
  </div>
```



```
<!-- 尾部脚本块 -->
<script type="text/javascript">
    $(function()
    {
        setTimeout(function()
        {
            $("a").click(function(event)
            {
                event.preventDefault(); alert(this.href);
            });
            $("span[rel^=href]").click(function(event)
            {
                alert(this.data);
            });
        }, 1000);
    });
</script>
</body>
</html>
```

上面的例子使用了两种方式实现模拟超链接的效果。

第一种还是使用<a>元素：

```
<a href="http://www.elong.com" onclick="return false;" target="_blank">超链接，点我！</a>
```

因为在标签上为元素添加了“onclick= ‘return false;’”语句，所以在前 1 秒内，单击此链接是没有任何反应的。这种方式的弊端很明显，这种在元素上绑定事件的方式不利于显示和行为的分离，并且没有处理事件的冒泡，没有实现多播事件等。但是对于搜索引擎是友好的，因为蜘蛛并不会真正地执行单击操作，<a>元素的目标页无论是否弹出都会被搜索引擎抓取到，是 SEO 友好的。

第二种方式是使用了<span>模拟一个超链接：

```
<span rel="href" data="http://www.elong.com" ><u style="cursor:pointer;">超链接，点我！</u></span>
```

为 span 添加了一个非标准的属性 data 用于存储数据，因为 span 没有类似 a 元素的默认行为，所以在 JavaScript 语句没有执行前单击同样没有任何效果。

## 3.3 jQuery 帮助文档

“授人以鱼，三餐之需；授人以渔，终生之用”。一本书讲解得再好，也不可能面面俱到。在很多开发人员眼里，看一本教材书不如直接查 API 文档学得快。jQuery 官网提供了详尽的各个 API 函数的文档及实例，所有学习 jQuery 及使用 jQuery 的人都要常去那里查找接口说明、接口使用例子等。本节将介绍 jQuery 在线文档的使用方式。

有人会想如果能在线学习，谁还需要看书呢？因为本书并不是 jQuery 官方文档的“中文翻译”，除了基本 jQuery 函数的使用，还用到更多的最佳实践和使用技巧，并且深入剖析 jQuery 的实现方式，这些都是“帮助手册”无法提供的。

### 3.3.1 jQuery API 在线帮助文档

jQuery 在线帮助文档是 jQuery 官网提供的，即使后面的中文帮助文件也都是在翻译







jQuery 在线帮助文档的基础上实现的。在 jQuery 1.4 版本发布后，在线 API 帮助文档已经改头换面，启用了新的网址，并且对函数的分类进行了重新调整。jQuery API 在线帮助文档：<http://api.jquery.com/>，如图 3-5 所示。

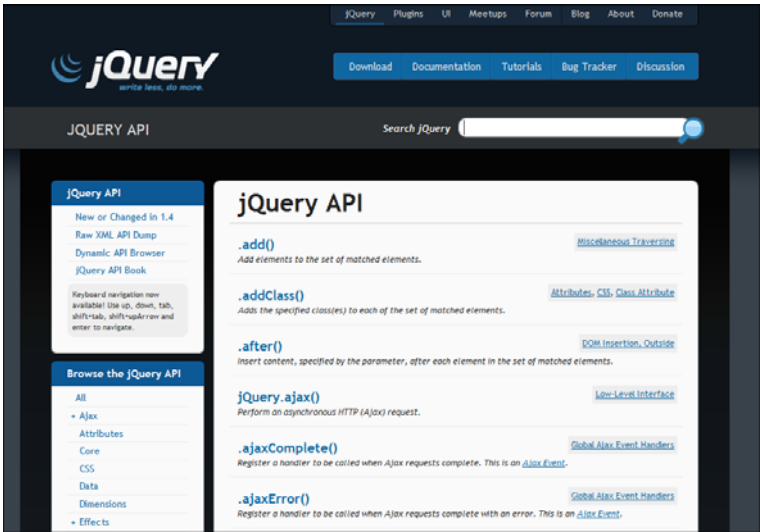


图 3-5 jQuery API 在线帮助文档

jQuery UI 在线帮助文档：<http://jqueryui.com/demos/>，如图 3-6 所示。

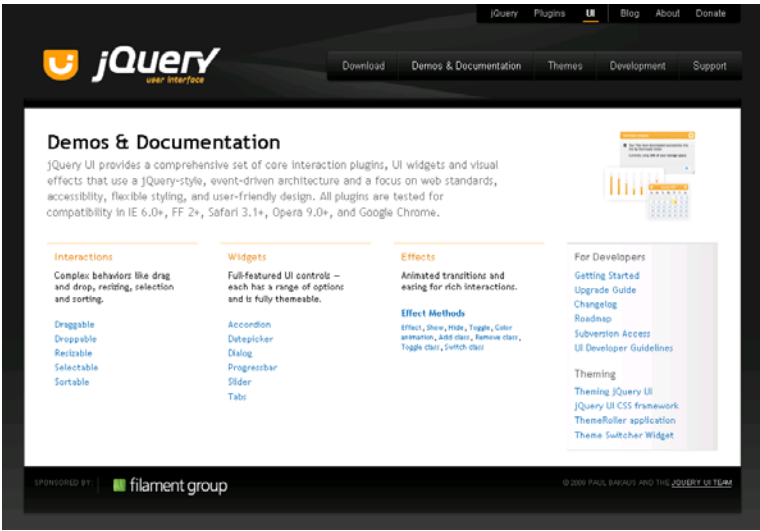


图 3-6 jQuery UI 在线帮助文档

jQueryAPI 在线帮助文档是开发 jQuery 最常使用的。jQuery UI 文档则是使用 jQuery UI 进行用户 UI 设计时使用的文档。在 jQuery UI 的章节将详细介绍 jQuery UI 库的使用。

### 3.3.2 jQuery API 在线帮助文档分类

jQuery API 在线帮助文档的“Browse the jQuery API”区域列出了 jQuery 的函数分类，如图 3-7 所示。





图 3-7 jQuery 函数分类

jQuery 的 API 拥有明确的分类，所以学习 jQuery 一定要注意 jQuery 的知识体系结构，这样才可以在文档中找到自己想要的 API 分类。

本书的第二部分将严格按照 jQuery 的知识体系讲解 jQuery。不可否认在进行每个分类 API 的讲解时，会略显无聊，但是打好基础及让知识体系清晰是十分有必要的。当学习完 jQuery 类库所有分类后，在本书的第三部分会开始 jQuery 的精彩案例及使用 jQuery UI 创造丰富的用户体验。

### 3.3.3 jQuery API 中文帮助文档

想必很多人手里都有一些 chm 格式的帮助手册。如果你的英文不好或者网速缓慢无法查询在线文档，“jQuery API 中文帮助文档”可以为你提供帮助。最新的 jQuery 中文帮助文档目前是 Google Code 上面的一个项目：<http://code.google.com/p/jquery-api-zh-cn/>，如图 3-8 所示。



图 3-8 jQuery API 中文帮助文档项目



这是由《锋利的 jQuery》一书的作者张晓菲（英文名 shawphy）创建的项目。在“Downloads”中可以下载到 chm 格式的中文手册，如图 3-9 所示。

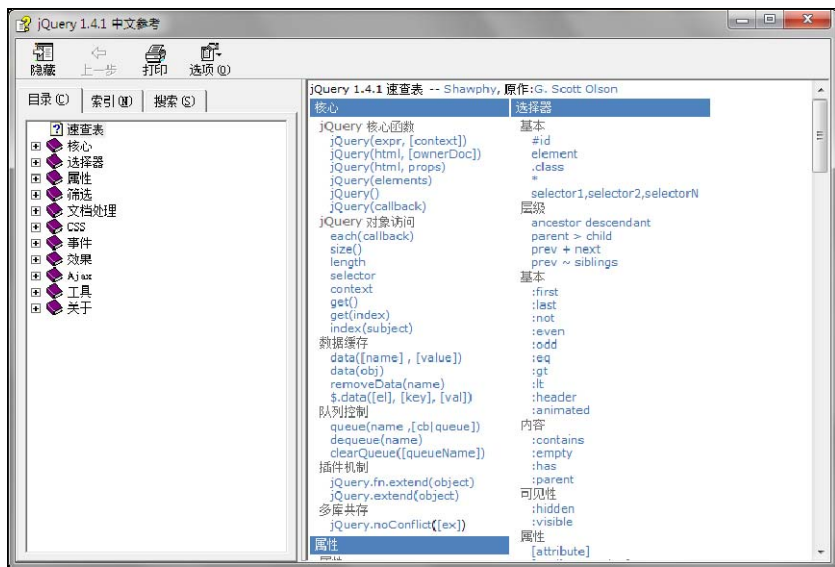


图 3-9 jQuery 中文手册

虽然目前此文档并不完善，里面还存在很多的错误，但是 jQuery API 中文文档的存在极大地方便了开发人员，尤其是里面还有很多的实例。相信随着不断的完善，此文档会成为 jQuery 开发人员的必备文档之一。

## 3.4 小结

本章讲解了 jQuery 的两部分核心知识：jQuery 对象和 jQuery 文档处理程序。在学习 jQuery 的功能函数之前，先学习这些 jQuery 的核心知识会使今后的学习事半功倍。另外本章还对 jQuery 的部分实现作了分析，即使以后不使用 jQuery 类库，也仍然可以从 jQuery 的实现中学到知识，这些原理性的知识会让 JavaScript 开发人员受用终身。

本章最后介绍了 jQuery 的在线文档手册和中文文档手册。介绍这部分内容是为了教大家如何自学 jQuery，在遇到问题时可以如何自己解决问题。

至此第一部分内容已经介绍完毕，第二部分将开始系统地学习各个分类的 jQuery API 函数。



## 第 4 章



# 万能的 jQuery 选择器

jQuery 选择器是 jQuery 类库提供的最强大的功能之一,是所有功能的基础。本章讲解 jQuery 最重要的选择器部分的知识。有了 jQuery 的选择器几乎可以获取页面上任意的一个或一组对象,可以明显减轻开发人员的工作量。



## 4.1 jQuery 选择器基础

本节将介绍 jQuery 选择器的基础知识。首先来了解一下何为 jQuery 选择器。

### 4.1.1 什么是 jQuery 选择器

使用 JavaScript 操作页面上的 DOM 元素时，首先要获取 DOM 元素。但是原始的 JavaScript 只元件根据 ID 或者 TagName 获取 Dom 对象。

在 jQuery 中则完全不同，jQuery 提供了异常强大的选择器用以帮助我们获取页面上的对象，并且将对象以 jQuery 对象的形式返回。

首先来看看什么是选择器。

```
//根据 ID 获取 jQuery 对象  
var jQueryObject = $("#testDiv");
```

上例中使用了 ID 选择器，选取 ID 为 testDiv 的 Dom 对象并将它放入 jQuery 对象，最后返回了一个 jQuery 对象。

现在通过 jQueryObject 变量就可以操作 testDiv 图层了，因为 jQueryObject 是一个 jQuery 对象，所以可以使用所有的 jQuery 对象方法。比如修改图层中的 HTML 内容：

```
jQueryObject.html("修改后的 HTML 内容")
```

通过 ID 选中元素是最有效率的 jQuery 选择器。这是因为在原始的 JavaScript 中就提供了选中 ID 的方法 document.getElementById()。有关选择器的使用技巧将在后面讲解。

jQuery 选择器的强大在于提供了丰富的选择器，如果使用原始 JavaScript 则需要编写很多的代码才能实现。这极大减少了开发人员的工作量。

### 4.1.2 jQuery 选择器核心函数

jQuery 选择器调用的是 jQuery 核心函数：

```
jQuery(expression, [context])
```

这个函数接收一个包含选择器表达式的字符串，然后用这个字符串去匹配一组元素。

jQuery 的核心功能都是通过这个函数实现的。jQuery 中的大部分功能都基于这个函数，或者说都是在以某种方式使用这个函数。该函数最基本的用法就是向它传递一个表达式（通常由 CSS 选择器组成），然后根据这个表达式查找所有匹配的元素。

默认情况下，如果没有指定 context 参数，\$()将在当前的 HTML document 中查找 DOM 元素；如果指定了 context 参数，如一个 DOM 元素集或 jQuery 对象，则会在这个 context 中查找。在 jQuery 1.3.2 以后，其返回的元素顺序等同于在 context 中出现的先后顺序。

参数说明如下。

- ☐ expression: 必选参数，选择器表达式。
- ☐ Context: 可选参数，选择器上下文。
- ☐ jQuery 选择器返回的是 jQuery 对象，jQuery 对象是一个集合，可以使用链式语法调用各种 jQuery 函数，这些在第 3 章有过详细的讲解。



- ❑ context 参数能够缩小选择器的查找范围，加快查找速度。在后面的 4.1.5 节将详细讲解如何使用 context。

4.1.3 jQuery 选择器分类

jQuery 选择器按照功能和类型分为下面几类，如表 4-1 所示。

表 4-1 选择器分类

名称名称	中文名称
Basics	基本选择器
Hierarchy	层级选择器
Basic Filters	基本过滤器
Content Filters	内容过滤器
Visibility Filters	可见性过滤器
Attribute Filters	属性选择器
Child Filters	子元素选择器
Forms	表单选择器
Form Filters	表单过滤器

jQuery 选择器可以分为两大类，“选择”和“过滤”。选择的作用是选择元素，过滤的作用是从选中的元素中筛选元素。如果只使用过滤器，相当于先选中所有元素然后过滤。比如属性过滤器：

```

$("[my=abc]").htmlAdd("/使用属性选择器");

```

等同于：

```

$("*[my=abc]").htmlAdd("/使用属性选择器(*)");

```

本节最后将用列表的形式对所有的选择器进行介绍，并对重要的选择器进行讲解。

4.1.4 使用 jQuery 选择器实验室

jQuery 选择器实验室是一个页面程序，可以在此页面上使用各种选择器，并查看应用效果。选择器实验室在本书代码的下面地址中：

```

【代码路径： jQueryStorm.Web /chapter4/ Demo-1-SelectorsLab.html 】

```

jQuery 选择器实验室分为以下三个区域。

- ❑ 选择器输入区域，如图 4-1 所示。

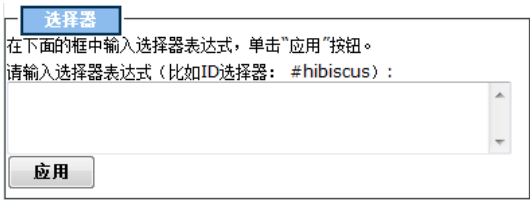


图 4-1 选择器输入区域

在此区域中，可以输入选择器表达式，单击“应用”按钮后将显示执行的 jQuery 语句及显示结果，如图 4-2 所示。



图 4-2 应用选择器表达式

□ DOM 元素区域，如图 4-3 所示。

DOM 元素区域是应用选择器的实验页面。在选择器区域输入了选择器表达式并且应用后，会对选中元素添加红色边框，如图 4-4 所示。

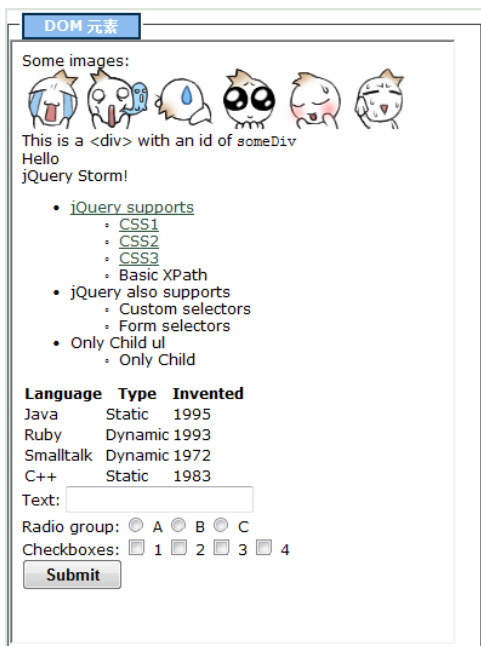


图 4-3 DOM 元素区域

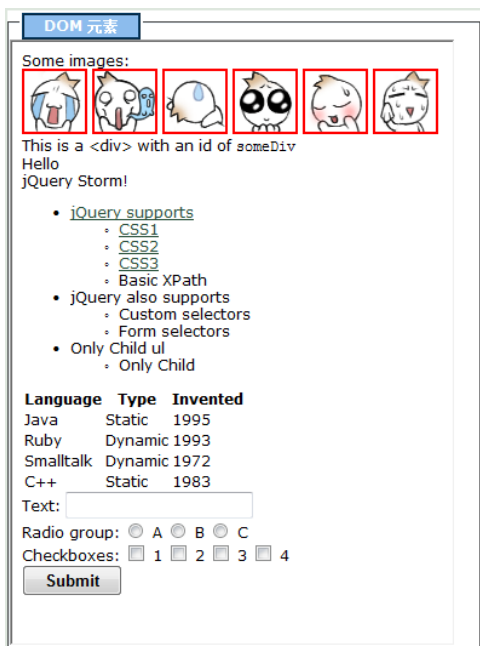


图 4-4 DOM 元素区域应用效果

□ DOM 代码区域，如图 4-5 所示。

Dom 代码区域显示 DOM 元素区域的 HTML 源代码。知道了源代码就可以快速地编写表达式选择页面元素。



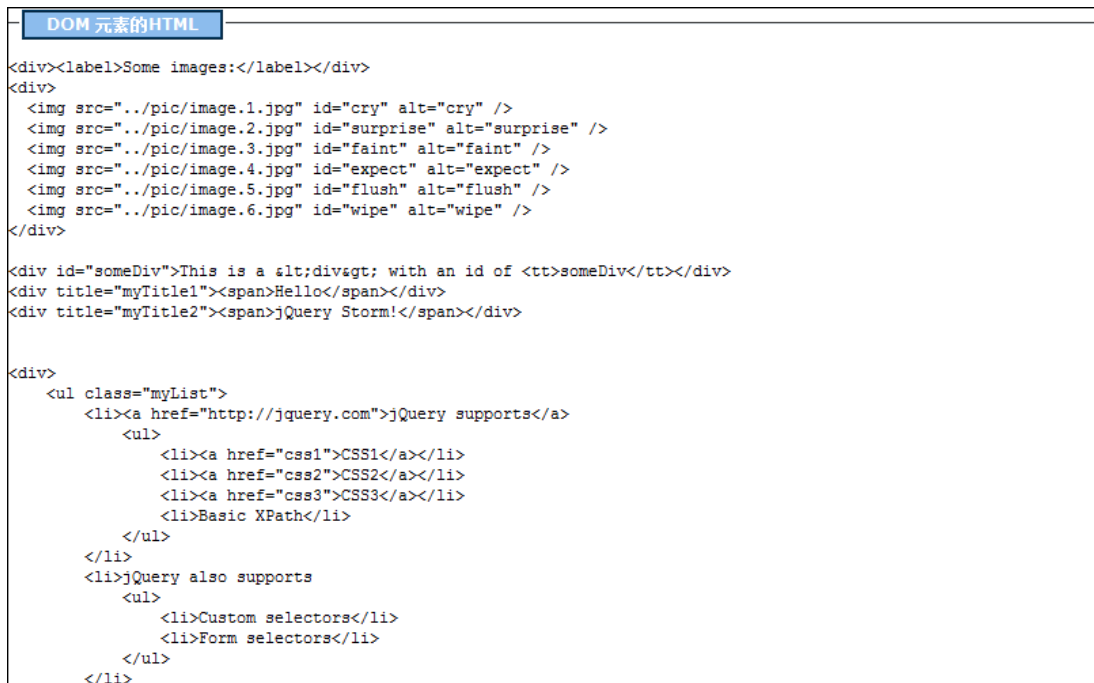


图 4-5 DOM 元素代码区域

有了选择器实验室，就可以快速地实验各种选择器，查看应用效果。如果觉得实验室的 DOM 页面内容无法满足某些选择器的实验条件，可以修改“data\SelectorsDomData.html”页面，选择器实验室的 DOM 元素和 DOM 代码都会随之更新。

#### 4.1.5 选择器使用技巧

- ❑ 选择器返回的是 jQuery 对象，jQuery 对象是一个集合，即使只选中了一个元素也是以集合的形式返回。
- ❑ 选择器中的特殊字符使用两个斜杠“\\”转义。
- ❑ 选择器即使没有选中页面元素，在 jQuery 对象上调用方法也不会报错，即永远不会返回 null。可以通过 size()方法取 jQuery 对象集合的个数来判断是否选中了元素。
- ❑ jQuery 对象的索引器返回的是 DOM 对象：

```
$("#img")[0];
```

不能在上面直接调用 jQuery 函数，可以使用 \$()函数将其转换成 jQuery 对象：

```
$("#img")[0].size();
```

- ❑ ID 选择器最快，因为 jQuery 内部使用 javascript 原生的 getElementById()方法获取元素。所以应该尽量从 ID 选择器开始，比如：

```
$("#myid").find(".className");
```

上面的代码速度要远大于

```
$(".className");
```

- ❑ 在无法使用 ID 选择器的前提下，优先从元素选择器开始。因为元素选择器使用





JavaScript 原生的 `getElementsByTagName()` 方法获取元素集合。所以速度仅次于 ID 选择器。优先使用元素选择器缩小查找集合，比如：

```
$("#input.className");
```

速度要优于：

```
$(".className")
```

□ 给选择器一个上下文。

在 `jQuery()` 选择器方法中实际上是调用内部的

```
return new jQuery.fn.init( selector, context );
```

方法。第二个参数 `context` 就是选择器上下文，即选择器查找的范围。不传递此参数则默认为 `document`。如果能缩小查找范围，则应该尽量传递 `context` 参数。但是，要注意 `$("#id").find(".className")` 与 `$("#id.className")` 的不同。

如果页面有如下结构：

```
<div id="id">
  <p class="className">Test</p>
</div>
```

使用下面两个语句都能够选择到 `p` 元素：

```
$("#id").find(".className");
```

或者

```
$("#id.className")
```

但是两者的实现方式却截然不同。

使用 `$("#id").find(".className")` 语句首先使用 ID 选择器选中 `div` 元素，然后使用：

```
jQuery.find(".className", div 元素, ret );
```

在 `div` 元素中查找样式为 “`className`” 的元素。

使用 `$("#id.className")` 语句则不同，会调用：

```
jQuery.find("#id.className", document 元素, ret );
```

这相当于从整个文档中查找，所以说速度应该慢一些。

但是在 1.3 及以后版本的 `jQuery` 中已经做了改进，如果在 IE 8、Firefox 等新生代浏览器中，会发现其实速度并不慢，因为当查找范围是 `document` 时会调用 `document` 的 `querySelectorAll()` 方法：

```
document.querySelectorAll("#id.className")
```

此方法的效率很高。但是鉴于在很长一段时间内，还会有很多用户使用 IE 6 等浏览器，所以应优先使用 `$("#id").find(".className")` 这种方式查找元素，因为这相当于给选择器指定了上下文。

□ 保存选择器返回的对象，减少选择器的使用次数。比如：

```
$(div).each(function(){
    $(this).click(function(){
        alert('click');
    });
});
```



```
});
$(this).find('span').css('color','red');
});
```

上面的代码使用了两次 jQuery 选择器 “\$(this)”，可以优化成：

```
$(div').each(function(){
    var _self=$(this);
    _self.click(function(){
        alert('click');
    });
    _self.find('span').css('color','red');
});
```

## 4.2 基础选择器

从本节开始，将用列表的形式介绍各个选择器，对于经常使用的选择器将重点举例讲解。首先学习基础选择器。基础选择器是最简单的选择器，其中比如 ID 选择器、元素选择器都是 JavaScript 原生支持的功能，所以在 jQuery 内部可以直接调用，效率最高。

### 4.2.1 基础选择器列表

如表 4-2 所示为所有的基础选择器。

表 4-2 基础选择器列表

名 称	中文常用叫法	说 明	举 例
#id	ID 选择器	根据元素 ID 选择	\$("#divId") 选择 ID 为 divId 的元素
element	元素选择器	根据元素的名称选择	\$("a") 选择所有<a>元素
.class	样式选择器	根据元素的 CSS 类选择	\$(".bgRed") 选择所用 CSS 类为 bgRed 的元素
*	全选择器	选择所有元素	\$("*")选择页面所有元素
selector1, selector2, selectorN	多重选择器	可以将几个选择器用“,”分隔开然后再拼成一个选择器字符串。会同时选中这几个选择器匹配的内容	\$("#divId, a, .bgRed")

### 4.2.2 基础选择器使用要点

基础选择器定义的都是最基础且最常使用的选择器。根据选择器优化原则，应该优先使用 ID 选择器和 element 选择器缩小查找范围。如果只是用过滤器，则相当于使用了 “\*” 选择器在所有元素中过滤，应该尽量避免。

## 4.3 层次选择器

层次选择器用于带有层次关系的对象选择。页面开发时经常遇到获取一个元素中的某些元素，或者获取相邻结点元素的使用场景，此时层次选择器就可以帮助我们完成任务。



### 4.3.1 层次选择器列表

如表 4-3 所示为所有的层次选择器。

表 4-3 层次选择器列表

名 称	常用中文叫法	说 明	举 例
ancestor descendant	后代选择器	使用“form input”的形式选中 form 中的所有 input 元素。即 ancestor(祖先)为 from, descendant(子孙)为 input	\$(".bgRed div") 选择 CSS 类为 bgRed 的元素中的所有<div>元素
parent > child	子代选择器	选择 parent 的直接子结点 child child 必须包含在 parent 中并且父类是 parent 元素	\$(".myList>li") 选择 CSS 类为 my List 元素中的直接子结点<li>对象
prev + next	层次选择器	prev 和 next 是两个同级别的元素。选中在 prev 元素后面的 next 元素	\$("#hibiscus+img")选在 ID 为 hibiscus 元素后面的 img 对象
prev ~ siblings	层次选择器	选择 prev 后面的根据 siblings 过滤的元素 注:siblings 是过滤器	\$("#someDiv~[title]")选择 ID 为 some Div 的对象后面所有带有 title 属性的元素

### 4.3.2 层次选择器使用要点

“ancestor descendant”后代选择器是最常用的选择器，但是要注意后代选择器和子代选择器的区别。很多时候使用后代选择器和子代选择器的结果是相同的，但是也有例外，比如选择器实验室中的下列 DOM 片段：

```
<form action="" method="put" onsubmit="return false;">
  <div>
    <label>Text:</label> <input type="text" id="aTextField" name="someTextField"/>
    <fieldset>
      <label>Newsletter:</label>
      <input name="newsletter" id="newsletter"/>
    </fieldset>
  </div>
  <div>
    <label>Radio group:</label>
    <input type="radio" name="radioGroup" id="radioA" value="A"/> A
    <input type="radio" name="radioGroup" id="radioB" value="B"/> B
    <input type="radio" name="radioGroup" id="radioC" value="C"/> C
  </div>
  <div>
    <label>Checkboxes:</label>
    <input type="checkbox" name="checkboxes" id="checkbox1" value="1"/> 1
    <input type="checkbox" name="checkboxes" id="checkbox2" value="2"/> 2
    <input type="checkbox" name="checkboxes" id="checkbox3" value="3"/> 3
    <input type="checkbox" name="checkboxes" id="checkbox4" value="4"/> 4
  </div>
  <button type="submit" id="submitButton">Submit</button>
</form>
```

代码中加粗的 ID 为“newsletter”的 input 元素。注意使用下面的选择器后此元素的选中状态。

使用后代选择器：

```
$("#div input")
```

结果为：



匹配的元素个数: 9  
INPUT#aTextField  
INPUT#newsletter  
INPUT#radioA  
INPUT#radioB  
INPUT#radioC  
INPUT#checkbox1  
INPUT#checkbox2  
INPUT#checkbox3  
INPUT#checkbox4

使用子代选择器:

`$("div>input")`

结果为:

匹配的元素个数: 8  
INPUT#aTextField  
INPUT#radioA  
INPUT#radioB  
INPUT#radioC  
INPUT#checkbox1  
INPUT#checkbox2  
INPUT#checkbox3  
INPUT#checkbox4

上述输出的区别在于, ID 为 newsletter 的 input 不是 div 的“直接子代”, 它的父类不是 div, 所以使用子代选择器时没有被选中。

4.4 基本过滤器

过滤器和选择器是有区别的。在开篇已经提到, 过滤器的作用是在已经选择的元素中进行“过滤”操作。因为单独使用过滤器, 相当于使用了“\*”这个全选择器, 性能会降低。所以在使用过滤器时, 一定要在过滤器前添加选择器。

4.4.1 基本过滤器列表

如表 4-4 所示为所有的过滤器。

表 4-4 基本过滤器列表

名 称	说 明	举 例
:first	匹配找到的第一个元素	查找表格的第一行: \$("tr:first")
:last	匹配找到的最后一个元素	查找表格的最后一行: \$("tr:last")
:not(selector)	去除所有与给定选择器匹配的元素 在 jQuery 1.3 中, 已经支持复杂选择器了 (例如: not(div a) 和 :not(div,a))	查找所有未选中的 input 元素: \$("input:not(:checked)")
:even	匹配所有索引值为偶数的元素, 从 0 开始计数	查找表格的 1、3、5...行: \$("tr:even")
:odd	匹配所有索引值为奇数的元素, 从 0 开始计数	查找表格的 2、4、6 行: \$("tr:odd")
:eq(index)	匹配一个给定索引值的元素 注:index 从 0 开始计数	查找第二行: \$("tr:eq(1)")



(续表)

名 称	说 明	举 例
:gt(index)	匹配所有大于给定索引值的元素 注:index 从 0 开始计数	查找第二行、第三行,即索引值是 1 和 2,也就是比 0 大:\$("#tr:gt(0)")
:lt(index)	选择结果集中索引小于 N 的 elements 注:index 从 0 开始计数	查找第一行、第二行,即索引值是 0 和 1,也就是比 2 小:\$("#tr:lt(2)")
:header	选择所有 h1,h2,h3 一类的 header 标签	给页面内所有标题加上背景色: \$(".header").css("background", "#EEE");
:animated	匹配所有正在执行动画效果的元素	只有对不在执行动画效果的元素执行一个动画特效: \$("#run").click(function(){ \$("#div:not(:animated)").animate({ left: "+=20" }, 1000); });

#### 4.4.2 基本过滤器使用要点

虽然称为“基本过滤器”,但是此分类中的过滤器使用的场景却很少。只有 not 过滤器最常被使用:

```
:not(selector)
```

not 过滤器能够从选中的元素中,排除一部分元素,比如选择器实验室的下列 HTML 片段:

```
<div id="someDiv">This is a &lt;div> with an id of <tt>someDiv</tt></div>  
<div title="myTitle1"><span>Hello</span></div>  
<div title="myTitle2"><span>jQuery Storm!</span></div>
```

如果想选中不包含“title”属性的 div,则可以使用如下选择器表达式:

```
div:not([title])
```

需要注意的是,在 1.3 版本以后的 jQuery 中,not 过滤器的 selector 参数已经支持复杂选择器了,比如后代选择器:

```
:not(div a)
```

多重选择器:

```
:not(div,a)
```

注意,not 过滤器的表达式是在选择出来的元素上应用的,比如上例中的:

```
div:not([title])
```

相当于首先使用 div[title]查找含有 title 的 div,然后在\$(div)找到的所有 div 中排除掉这些 div。



## 4.5 内容过滤器

### 4.5.1 内容过滤器列表

如表 4-5 所示为所有的内容过滤器。

表 4-5 内容过滤器列表

名 称	说 明	举 例
:contains(text)	匹配包含给定文本的元素	查找所有包含“John”的 div 元素:\$("#div:contains('John')")
:empty	匹配所有不包含子元素或者文本的空元素	查找所有不包含子元素或者文本的空元素:\$("#td:empty")
:has(selector)	匹配含有选择器所匹配的元素	给所有包含 p 元素的 div 元素添加一个 text 类: \$("#div:has(p)").addClass("test")
:parent	匹配含有子元素或者文本的元素	查找所有含有子元素或者文本的 td 元素:\$("#td:parent")

### 4.5.2 内容过滤器使用要点

内容过滤器中的 contains 过滤器和 has 过滤器都是经常使用的过滤器。一个用来查找包含“指定文本”的元素，一个用来查找包含“指定元素”的元素。

contains 匹配的是文本，注意如果文本包含在元素的子元素中，contains 同样认为是符合条件的，比如选择器实验室的下列 HTML 片段：

```
<div>
  <ul class="myList">
    <li><a href="http://jquery.com">jQuery supports</a>
      <ul>
        <li><a href="css1">CSS1</a></li>
      </ul>
    </li>
  </ul>
</div>
```

使用表达式：

```
div:contains("CSS1")
```

同样可以选中此 div。因为 div 的子元素中包含了 css1 文本。

has 过滤器用来查找“匹配含有选择器所匹配的元素”，和 not 过滤器不同，has 过滤器是在已经选中的元素内部查找子元素，比如：

```
$(div:has(img));
```

可以将 not 过滤器和 has 过滤器组合使用。比如想要查找“不含有 img 元素的 div”：

```
$(div:not(:has(img)));
```

## 4.6 可见性过滤器

### 4.6.1 可见性过滤器列表

如表 4-6 所示为所有的可见性过滤器。





表 4-6 可见性过滤器列表

名 称	说 明	举 例
:hidden	匹配所有的不可见元素 注:在 1.3.2 版本中, hidden 匹配自身或者父类在文档中不占用空间的元素。 如果使用 CSS visibility 属性让其不显示但是占位, 则不属于 hidden.	查找所有 display 为 none 的 tr 元素:\$(“tr:hidden”)
:visible	匹配所有的可见元素	查找所有 display 不为 none 的 tr 元素:\$(“tr:visible”)

## 4.6.2 可见性过滤器使用要点

可见性过滤器根据元素的 display 属性选择元素, 特别需要注意的是 CSS 的 visibility 属性也能控制元素的实现。visibility 为 hidden 的元素也会隐藏, 但是仍然占据页面控件, 这种元素符合 “:visible”, 而不是 “:hidden”。可以理解为 visibility 样式与可见性过滤器无关。

比如选择器实验室的下面两个 tr 片段:

```
<tr id="display_none" style="display:none;">
  <td>Smalltalk</td>
  <td>Dynamic</td>
  <td>1972</td>
</tr>
<tr id="visibility_hidden" style="visibility:hidden;">
  <td>C++</td>
  <td>Static</td>
  <td>1983</td>
</tr>
```

则两个 tr 都是不可见的, 但是 ID 为 visibility\_hidden 的 tr 仍然占据着页面控件。使用可见性选择器的示例如下:

```
$(“tr:hidden”);//选中 ID 为 display_none 的元素
$(“tr:visible”);//选中 ID 为 visibility_hidden 的元素
```

## 4.7 属性过滤器

### 4.7.1 属性过滤器列表

如表 4-7 所示为所有的属性过滤器。

表 4-7 属性过滤器列表

名 称	说 明	举 例
[attribute]	匹配包含给定属性的元素	查找所有含有 ID 属性的 div 元素: \$(“div[id]”)
[attribute=value]	匹配给定的属性是某个特定值的元素	查找所有 name 属性是 newsletter 的 input 元素: \$(“input[name=‘newsletter’]”).attr(“checked”, true)
[attribute!=value]	匹配给定的属性是不包含某个特定值的元素	查找所有 name 属性不是 newsletter 的 input 元素: \$(“input[name!=‘newsletter’]”).attr(“checked”, true)
[attribute^=value]	匹配给定的属性是以某些值开始的元素	\$(“input[name^=‘news’]”)

(续表)

名 称	说 明	举 例
[attribute\$=value]	匹配给定的属性是以某些值结尾的元素	查找所有 name 以 'letter' 结尾的 input 元素: \$("input[name\$='letter']")
[attribute*=value]	匹配给定的属性是以包含某些值的元素	查找所有 name 包含 'man' 的 input 元素: \$("input[name*='man']")
[attributeFilter1][attributeFilter2] [attributeFilterN]	复合属性选择器, 需要同时满足多个条件时使用	找到所有含有 ID 属性, 并且它的 name 属性是以 man 结尾的: \$("input[id][name\$='man']")

4.7.2 属性过滤器使用要点

属性过滤器是最常使用的过滤器，使用属性过滤器甚至可以完成许多其他过滤器的功能，比如可以替代样式选择器：

```
//下面两条语句具有相同的功能
$(".myList");
$("[class=myList]");
```

可以替代表单类别过滤器：

```
//下面两条语句具有相同的功能
$("input[type=radio]");
$(":radio");
```

所以如果记不住花样繁多的 jQuery 选择器，那么记住基础选择器和属性过滤器就可以完成大部分的任务。

在使用属性过滤器时尤其要注意，尽量在过滤前缩小范围，否则默认 “\*” 在所有元素中查找元素。比如：

```
$(".myList"); //速度最慢，等同于$("*[class=myList]");
$("div[class=myList]"); //速度较快
$("#id div[class=myList]"); //速度最快
```

4.8 子元素过滤器

子元素过滤器用于从子对象集合中进行过滤。

4.8.1 子元素过滤器列表

如表 4-8 所示为所有的子元素过滤器。

表 4-8 子元素过滤器列表

名 称	说 明	举 例
:nth-child(index/even/odd/equation)	匹配其父元素下的第 N 个子或奇偶元素  'eq(index)' 只匹配一个元素，而这个将为每一个父元素匹配子元素。:nth-child 从 1 开始的，而:eq()是从 0 算起的！ 可以使用：	在每个 ul 查找第 2 个 li: \$("ul li:nth-child(2)")

(续表)

名 称	说 明	举 例
:nth-child(index/even/odd/equation)	nth-child(even) :nth-child(odd) :nth-child(3n) :nth-child(2) :nth-child(3n+1) :nth-child(3n+2)	
:first-child	匹配第一个子元素 'first' 只匹配一个元素，而此选择符将为每个父元素匹配一个子元素	在每个 ul 中查找第一个 li: \$("ul li:first-child")
:last-child	匹配最后一个子元素 'last' 只匹配一个元素，而此选择符将为每个父元素匹配一个子元素	在每个 ul 中查找最后一个 li: \$("ul li:last-child")
:only-child	如果某个元素是父元素中唯一的子元素，那将会被匹配 如果父元素中含有其他元素，那将不会被匹配	在 ul 中查找是唯一子元素的 li: \$("ul li:only-child")

#### 4.8.2 子元素过滤器使用要点

子元素过滤器并不经常使用，因为通常需要过滤的是具有某些业务逻辑的对象。使用时需要注意 nth-child 过滤器是从 1 开始的，而 eq 过滤器是从 0 开始的。

### 4.9 表单类别过滤器

此分类在 jQuery 官方的分类是“Form”，而将“表单内容过滤器”称之为“Form Filter”。本书将 Form 有关的选择器分为两部分：“表单类别过滤器”和“表单属性过滤器”。顾名思义，一个是根据类别过滤表单元素。另一个是根据属性值的状态过滤表单元素。

#### 4.9.1 表单类别过滤器列表

如表 4-9 所示为用于表单类别过滤的过滤器。

表 4-9 表单类别过滤器列表

名 称	说 明	解 释
:input	匹配所有 input、textarea、select 和 button 元素	查找所有的 form 元素: \$("input")
:text	匹配所有的文本框	查找所有文本框: \$("text")
:password	匹配所有密码框	查找所有密码框: \$("password")
:radio	匹配所有单选按钮	查找所有单选按钮
:checkbox	匹配所有复选框	查找所有复选框: \$("checkbox")

(续表)

名 称	说 明	解 释
:submit	匹配所有提交按钮	查找所有提交按钮: \$(":submit")
:image	匹配所有图像域	匹配所有图像域: \$(":image")
:reset	匹配所有重置按钮	查找所有重置按钮: \$(":reset")
:button	匹配所有按钮	查找所有按钮: \$(":button")
:file	匹配所有文件域	查找所有文件域: \$(":file")

4.9.2 表单类别过滤器使用要点

首先来看 form 中可能出现的表单元素：

```
<input type="text" value="Text" />
<input type="password" value="password" />
<input type="radio" value="radio" />
<input type="checkbox" value="checkbox" />
<input type="submit" value="submit" />
<input type="image" value="image" />
<input type="reset" value="reset" />
<input type="button" value="button" />
<button>Button Element</button>
<input type="file" value="file" />
<select>
  <option>select</option>
</select>
<textarea cols="10">textarea</textarea>
```

除了 input 过滤器,可以发现几乎每一个表单类别过滤器都对应一个 input 元素的 type 值。比如，下面两个选择器可以实现相同的功能：

```
$("input[type=image]");
$(":image");
```

大部分表单类别过滤器可以使用属性过滤器替换。但是因为 form 中的表单元素除了 input，还有几个特殊的元素 button、select、textarea。

可以使用 element 元素选择器选中这些元素，比如选中所有的 textarea 元素：

```
$("textarea");
```

但是如果希望选中所有的表单元素，就十分麻烦：

```
$("input,textarea,select,button");
```

可以使用 input 过滤器替代：

```
$("*:input");
```

其中 “\*” 可以省略。

具有同样特殊效果的还有 button 过滤器，下面两句话都可以选中页面上所有的 button。



```
$(":button");  
$("button,input[type='button']"); //等同于$(":button");
```

注意这里的 **button** 指的是表单元素的类型，而不是表现形式。在页面上 **submit**、**reset** 等表单元素也是以“按钮”的形式展现的。不同之处是 **submit** 有默认的浏览器行为提交表单，**reset** 有默认的浏览器行为重置表单。同样具有“浏览器默认行为”的还有 **a** 元素（默认打开 **href** 指定的链接）。可以通过下列代码取消元素浏览器的默认行为：

```
$(":submit").click(function(event){ event.preventDefault(); });
```

所以可以将 **button** 类别理解为“没有浏览器默认行为的表单元素”。

## 4.10 表单属性过滤器

表单属性过滤器是根据表单的属性值，比如“可用”、“不可用”、“选中”等状态值，对选中的集合进行过滤。

### 4.10.1 表单属性过滤器列表

如表 4-10 所示为所有的表单属性过滤器。

表 4-10 表单属性过滤器列表

名 称	说 明	解 释
:enabled	匹配所有可用元素	查找所有可用的 input 元素: \$("input:enabled")
:disabled	匹配所有不可用元素	查找所有不可用的 input 元素: \$("input:disabled")
:checked	匹配所有选中的被选中元素（复选框、单选框等，包括 select 中的 option）	查找所有选中的复选框元素: \$("input:checked")
:selected	匹配所有选中的 option 元素	查找所有选中选项的元素: \$("select option:selected")

### 4.10.2 表单属性过滤器使用要点

表单属性过滤器能够快速选中页面上特殊属性值的表单元素，这是 jQuery 最有用的选择器之一。

**checked** 过滤器可以查找选中的表单元素，虽然官方文档上说只会选中复选框 **checkbox** 和单选框 **radio**，但是实际上会连同 **option** 一起选中，比如：

```
$(":checked");
```

在选择器实验室中，返回结果：

```
应用的 jQuery 语句: $(":checked").addClass("wrappedElement")  
匹配的元素个数: 2  
INPUT  
OPTION
```



在 1.3.2 版本中，select 元素中被选中的 option 也会被选择出来。  
所以如果希望选择单选框和复选框，可以添加 input 元素选择器前缀：

```
$("#input:checked")
```

这样就可以排除 option 元素。

## 4.11 小结

本章介绍了 jQuery 最关键的选择器，将选择器按照功能分为“选择”和“过滤”两部分，使用列表的形式对所有的选择器进行了讲解，并在接下来的使用要点中对注意事项和重要的选择器进行了重点讲解，通过书中附带的 jQuery 选择器实验室可以方便地实验各种选择器。

其中 4.1.5 节是对 jQuery 选择器的最佳实践总结，需要仔细地学习并理解。





## 第 5 章



# 管理 jQuery 对象集合

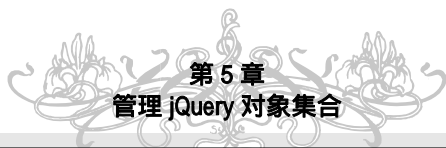
页面上也常会涉及创建 DOM 元素、删除 DOM 元素等操作。因为 jQuery 对象是以集合的形式存在的，所以还要执行将 DOM 对象添加到 jQuery 对象集合、从 jQuery 对象集合中删除等操作。jQuery 官方网站将管理 jQuery 对象集合的函数放在了“Traversing”分类中。“Traversing”分类下主要包括三个子类，如表 5-1 所示。

表 5-1 Traversing 函数分类

分类名称	中文名称
Filtering	过滤
Finding	查找
Chaining	链式操作

本文先讲解如何创建 DOM 元素，接着讲解“Traversing”分类下面的各个子类函数。





## 5.1 动态创建元素

### 5.1.1 使用 JavaScript 创建对象

在第2章中，介绍了 DOM 元素与 HTML 元素。通过 JavaScript 可以方便地获取一个 DOM 元素。

```
document.getElementById("divHolder");
```

通过 `getElementById()`、`getElementsByName()`、`getElementsByTagName()` 三个 JavaScript 函数，可以获取到 DOM 对象。

如果页面上希望动态地创建 DOM 对象并在页面上呈现，通常有两种方法：使用“innerHTML”或者标准的 DOM 方法。

假设页面上有如下元素：

```
<div id="divHolder"></div>
```

根据页面脚本框架创建当前页面的脚本对象，在 `initializeDom` 中获取上面的 div 容器对象：

```
var thisPage = {  
  initialize: function() { //加载时执行  
    this.initializeDom();  
    this.initializeEvent();  
  },  
  initializeDom: function() { //初始化 DOM  
    this.holder = document.getElementById("divHolder");  
  },  
  initializeEvent: function() { //事件绑定  
  },  
}
```

下面是使用 JavaScript 创建对象的两种方法。

(1) 使用 `innerHTML()` 方法。

```
var startDate = new Date();  
for (var i = 0; i < 100; i++) {  
  var str = "<div style='width:100px; height:20px;background-color:#eee'>test</div>";  
  this.holder.innerHTML = str;  
};  
this.holder.innerHTML += "使用 innerHTML 耗时: " + (new Date() - startDate);
```

(2) 使用标准 DOM 方法。

```
var startDate = new Date();  
for (var i = 0; i < 100; i++) {  
  var oDiv = document.createElement("div");  
  var oText = document.createTextNode("text");  
  oDiv.appendChild(oText);  
  this.holder.appendChild(oDiv);  
  oDiv.style.width = "100px";  
  oDiv.style.height = "20px";  
  oDiv.style.backgroundColor = "#eee";  
};  
this.holder.innerHTML += "<br/>使用 createElement 耗时: " + (new Date() - startDate);
```

使用 `innerHTML()` 方法和 `createElement()` 方法都可以创建新的对象，并让浏览器重新



解析 DOM。单从创建元素的速度上，innerHTML 在各个浏览器中的效率与 createElement 几乎相同。比如上面的代码，循环 100 次的效率在 IE8 下 innerHTML 能快几毫秒，在 Chrome 中则慢几毫秒。

但是 createElement() 是标准的 DOM 方法，也就是不仅仅可以用来操作 HTML，也可以用来操作 XML 文档等所有支持 DOM 操作的对象。innerHTML 的效率与如何使用有关，在 JavaScript 中最耗时的是字符串操作。实例代码为了减少字符串拼接造成的性能损失，使用 innerHTML 创建的对象，都用“=”操作而不是用“+=”添加到容器里：

```
this.holder.innerHTML = str;
```

这样做导致页面上将只会保留最后一次循环添加的元素，但还是执行了 100 次 innerHTML() 方法，所以如果考虑性能测试是比较公平的。

如果改成：

```
this.holder.innerHTML += str;
```

则实现 100 次循环的时间在 IE 8 下由 18 毫秒上升到 290 毫秒。可见字符串操作在 JavaScript 中消耗了相当多的时间。可以使用 Array 对象减少字符串操作：

```
createDom1: function() {
    var startDate = new Date();
    var builder = new Array();
    for (var i = 0; i < 100; i++) {
        builder.push("<div style='width:100px; height:20px;background-color:#eee>test</div>");
    };
    this.holder.innerHTML = builder.join("");
    this.holder.innerHTML += "使用 innerHTML 耗时: " + (new Date() - startDate);
}
```

这样做耗时只有 6 毫秒，因为除了减少了字符串操作，只调用了一次 innerHTML() 方法。

由此可见，当需要批量插入多个元素，并且可以在插入前构造完整的 HTML 代码时，使用 innerHTML 一次插入所有元素可以极大地提高效率。如果需要分别插入元素则使用 createElement() 方法更有效率。

了解了原始 JavaScript 创建对象的效率问题，有助于选择正确的 jQuery 方法。

### 5.1.2 使用 jQuery 创建对象

在 jQuery 中创建对象，需要使用 jQuery 核心函数，如表 5-2 所示。

表 5-2 jQuery 创建对象核心函数

名 称	类 型	说 明
jQuery( html, ownerDocument )	function	根据提供的原始 HTML 标记字符串，动态创建 DOM 元素。 最简单的不包含属性的元素，比如“<div>”将使用 document.createElement 创建，其他格式的字符串将使用 innerHTML 创建。HTML 字符串不能创建如 html、head、body、title 等无法放在 div 中的元素。 所有的字符串需要有正确的格式，否则在某些浏览器下可能无法正常工作。“<span>”可能无法正常工作，但是“<span/>”就可以正常工作（注意 XML 中的斜杠关闭符）
返回值	jQuery	新创建元素的引用，并以 jQuery 对象的形式返回

(续表)

名 称	类 型	说 明
html 参数	string	要创建的 HTML 字符串
ownerDocument 参数	document (可选)	创建 DOM 元素所在的文档

jQuery( html, ownerDocument ) 函数用来创建 DOM 元素，并且以 jQuery 对象的形式返回：

```
jQuery("<span/>").attr("id", "mySpan");
```

所以可以使用链式操作来操作 DOM 对象的属性和样式。

jQuery 内部根据 HTML 字符串的不同来决定使用何种方式创建元素。

对于不带属性的简单字符串，比如 “<div/>”，将使用 document.createElement 创建元素。

对于带属性的字符串，将使用 innerHTML 的方式创建元素。有趣的是，即使使用 innerHTML 的方式创建元素，在内部也会首先使用 document.createElement 创建一个 div 元素，然后调用 div 元素的 innerHTML 方法。下面代码中的 elem 就是传入的 HTML 字符串：

```
div = context.createElement("div");
div.innerHTML = wrap[1] + elem + wrap[2];
```

然后返回的是此 div 的子结点。

```
jQuery.makeArray(div.childNodes);
```

鉴于 innerHTML 和 createElement 的效率，如果要一次创建多个元素，首选 innerHTML 方法。

```
jQuery("<span>test</span><span>test</span><span>test</span>").appendTo(this.holder);
```

需要注意，HTML 字符串格式必须是闭合的，比如 “<span/>” 或者 “<span><span>”，但不能是

```
“<span>”。
```

### 5.1.3 创建对象常见错误

脚本开发人员要避免在 DOM 未加载完成时，改变 DOM 结构。比如在 body 中，加入了下面的语句：

```
jQuery("<div>test</div>").appendTo("body");
```

当页面较大或者网速较慢导致加载 DOM 缓慢时，就会出现“浏览器中止操作”的错误。使用第 3 章中介绍的 jQuery 文档处理程序，就可以解决此问题：

```
$(function{
    jQuery("<div>test</div>").appendTo("body");
});
```

上面的代码就能保证 JavaScript 语句在 DOM 加载完毕后执行。

另外，虽然不能使用 innerHTML 和 createElement（jQuery 内部也应用这两个方法所以同样不能使用）在页面加载的时候创建 DOM 元素，但是可以使用 document.write()方法。



```
document.write("<div>test</div>");
```

document.write()方法在页面加载时改变 DOM 结构，目前不会引发“浏览器终止”错误，但还是应该尽量避免使用。使用 jQuery 文档处理程序是最佳的方法。

## 5.2 过滤函数——筛选对象集合

Traversing 中的过滤函数对应于 jQuery 官方文档中的 Traversing-Filtering 分类。过滤函数的作用是在已经选定的集合中，将匹配过滤函数的元素保留，将不符合的去除。相关的过滤函数如表 5-3 所示。

### 5.2.1 过滤函数列表

表 5-3 过滤函数列表

名 称	说 明	举 例
eq( index )	返回集合中指定索引 index 的元素，从 0 开始计算	获取匹配的第二个元素： \$("p").eq(1)
filter( expr )	筛选出与指定表达式匹配的元素集合	保留带有 selected 类的元素： \$("p").filter(".selected")
filter( fn )	筛选出与指定函数返回值匹配的元素集合， 这个函数内部将对每个对象计算一次（正如 '\$.each'）。 如果调用的函数返回 false 则这个元素被删除，否则就会保留	保留子元素中不含有 ol 的元素： \$("div").filter(function(index) { return \$("ol", this).size() == 0; });
is( expr ) 注意：这个函数返回的不是 jQuery 包装集而是 Boolean 值	用一个表达式检查当前选择的元素集合，如果其中至少有一个元素符合这个给定的表达式就返回 true。 如果没有元素符合，或者表达式无效，则返回 false。 filter 内部实际也是在调用这个函数，所以，filter()函数原有的规则在这里也适用	由于 input 元素的父元素是一个表单元素， 所以返回 true： \$("input[type='checkbox']").parent().is("form")
map( callback )	将一组元素转换成其他数组（不论是否是元素数组） 可以用这个函数建立一个列表，不论是值、属性还是 CSS 样式，或者其他特别形式，都可以用 '\$.map()'来方便地建立	把 form 中的每个 input 元素的值建立一个列表： \$("p").append( \$("input").map(function(){ return \$(this).val(); }).get().join(", "));
not( expr )	删除与指定表达式匹配的元素	从 p 元素中删除带有 select 的 ID 的元素： \$("p").not( \$("#selected")[0] )
slice( start, end )	选取一个匹配的子集	选择第一个 p 元素： \$("p").slice(0, 1);

### 5.2.2 过滤函数要点

#### 1. Eq()函数

Eq()函数返回匹配集合中指定索引的元素，索引 index 是从 0 开始计算的。如果 index 超出了集合则返回一个空集合，而不会返回 null。

## 2. filter()函数与 find()函数

filter()函数支持选择器表达式和 fn()函数两种类型的参数，是最常使用的过滤函数。但是初学者常常分不清 filter()函数和后面即将讲到的 find()函数。其实在使用 jQuery 后代选择器时已经在内部使用过 find()函数：

```
jQuery("div.className");
```

等同于：

```
jQuery("div").find(".className");
```

使用 filter()和 find()都要首先使用选择器获得一个 jQuery 对象集合。

filter()函数是作用在集合的每一个对象上，即在“jQuery("div")”选取的集合上过滤，将匹配表达式或者函数的对象保留。

find()函数是在每一个对象内部查找匹配表达式的子元素，即作用在“jQuery("div")”集合的每一个子元素上。返回的是匹配的子元素集合。

下面的例子可以很好地解释这两个函数的差别，HTML 元素代码如下：

```
<div>
  <p class="testClass">测试 1</p>
</div>
<br />
<div class="testClass">
  <p class="testClass">测试 2</p>
</div>
```

为测试元素添加一些样式：

```
<style type="text/css">
div { border:solid 2px #00FF00;}
p {border: solid 2px #0000FF;}
</style>
```

效果如图 5-1 所示。

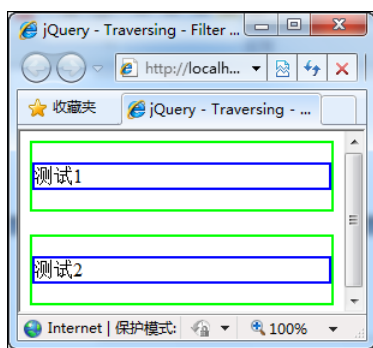


图 5-1 find()与 filter()函数差别实例

使用 filter()函数：

```
$("div").filter(".testClass").css("border", "solid 2px #FF0000");
```

效果如图 5-2 所示，第二个 div 边框变成了红色。

使用 find()函数：



```
$("#div").find(".testClass").css("border-color", "#FF0000");
```

效果如图 5-3 所示，两个 div 中的子元素 p 都被选中了。

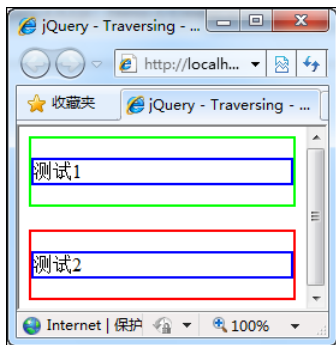


图 5-2 使用 filter()函数

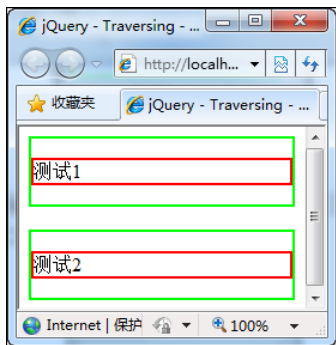


图 5-3 使用 find()函数

### 3. Is()函数

Is()函数返回的不是 jQuery 对象集合，而是 true 或者 false。只要 jQuery 对象集合中有一个元素满足表达式的条件，就返回 true。比如：

```
alert($("#div").is(".testClass"));
```

只要有一个 div 应用了“testClass”样式，则返回的是 true，否则是 false。

### 4. map()函数

map()函数会改变 jQuery 对象集合，比如：

```
$("#div").map(function() { return this.innerHTML; })
```

开始 jQuery 对象集合是两个 div 元素，使用 map 以后，jQuery 对象集合变成了两个 string，string 的内容就是 innerHTML 的内容：

```
<p class="testClass">测试 1</p> 和 <p class="testClass">测试 2</p>。
```

此时虽然仍然是以 jQuery 对象的形式返回，但因为是字符串所以无法执行 CSS、attr 等 DOM 对象操作了。另外在 map()函数中的 this 是 DOM 对象，如果要应用 jQuery 函数需要使用“\$(this)”。

### 5. slice()函数

slice()函数的行为与 JavaScript 数组的 slice()函数相同。即参数 start 表示其元素的索引，从 0 开始。

end 参数可选，如果不传入 end 参数则表示选取从 start 以后所有的元素。

## 5.3 查找函数——找到目标对象

查找函数的作用是从集合内再次查找匹配的元素，查找函数的清单如表 5-4 所示。过滤和查找这两类函数的作用并没有特别明显的切割，有时是可以彼此替换的。但是某些看似相同功能的函数还是有区别的，比如 5.2 节讲的 filter()函数和 find()函数的区别。

## 5.3.1 查找函数列表

表 5-4 查找函数列表

名 称	说 明	举 例
add( expr )	expr 可以是选择器表达式、DOM 对象、jQuery 对象、集合或者 HTML 字符串。如果是 HTML 字符串则会动态创建对象并添加到 jQuery 对象集合中	动态生成一个元素并添加至匹配的元素中: \$("p").add("<span>Again</span>")
children( [expr] )	取得 jQuery 对象集合中所有匹配满足表达式的直接子元素。 省略 expr 参数表示选择所有直接子元素。 注意: parents()将查找所有祖辈元素, 而 children()只考虑子元素而不考虑所有后代元素	查找 div 中的每个子元素: \$("div").children()
closest( [expr] )	jQuery 1.3 新增。从元素本身开始, 逐级向上级元素匹配, 并返回最先匹配的元素。 closest 会首先检查当前元素是否匹配, 如果匹配则直接返回元素本身。如果不匹配则向上查找父元素, 一层一层往上, 直到找到匹配选择器的元素。如果什么都没找到则返回一个空的 jQuery 对象。 closest 对于处理事件委派非常有用	为事件源最近的父类 li 对象更换样式: \$(document).bind("click", function (e) { \$(e.target).closest("li").toggleClass("highlight"); });
contents( )	查找匹配元素内部所有的子结点 (包括文本结点)。 如果元素是一个 iframe, 则查找文档内容	查找所有文本结点并加粗: \$("p").contents().not("[nodeType=1]").wrap("<b/>");
find( expr )	搜索所有与指定表达式匹配的元素。这个函数是找出正在处理元素的后代元素的好方法。 所有搜索都依靠 jQuery 表达式来完成。这个表达式可以使用 CSS1-3 的选择器语法来写	从所有的段落开始, 进一步搜索下面的 span 元素。 与\$("p span")相同: \$("p").find("span")
next( [expr] )	取得一个包含匹配的元素集合中每一个元素紧邻的后面同辈元素的元素集合。 这个函数只返回后面那个紧邻的同辈元素, 而不是后面所有的同辈元素 (可以使用 nextAll)。可以用一个可选的表达式进行筛选	找到每个段落后面紧邻的同辈元素: \$("p").next()
nextAll( [expr] )	查找当前元素之后所有的同辈元素。 可以用表达式过滤	给第一个 div 之后的所有元素加个类: \$("div:first").nextAll().addClass("after");
offsetParent( )	返回第一个有定位的父类 (比如 relative 或 absolute)	
parent( [expr] )	取得一个包含着所有匹配元素的唯一父元素的元素集合。 可以使用可选的表达式来筛选	查找每个段落的父元素: \$("p").parent()
parents( [expr] )	取得一个包含着所有匹配元素的祖先元素的元素集合 (不含根元素)。可以通过一个可选的表达式进行筛选	找到每个 span 元素的所有祖先元素: \$("span").parents()
prev( [expr] )	取得一个包含匹配的元素集合中每一个元素紧邻的前一个同辈元素的元素集合 可以用一个可选的表达式进行筛选。只有紧邻的同辈元素会被匹配到, 而不是前面所有的同辈元素	找到每个段落紧邻的前一个同辈元素: \$("p").prev()





(续表)

名 称	说 明	举 例
prevAll( [expr] )	查找当前元素之前所有的同辈元素 可以用表达式过滤	给最后一个之前的所有 div 加上一个类: \$("div:last").prevAll().addClass("before");
siblings( [expr] )	取得一个包含匹配的元素集合中每一个元素的所有 唯一同辈元素的元素集合。可以用可选的表达式进 行筛选	找到每个 div 的所有同辈元素: \$("div").siblings()

### 5.3.2 查找函数要点

#### 1. add()函数

add()函数是常用的函数,用于向 jQuery 对象集合中添加元素。add()函数的参数 expr 还可以传入 HTML 字符串来动态地创建元素,创建后的元素也将被添加到 jQuery 对象集合中。

#### 2. children()函数

children()函数可以省略参数,表示获取所有匹配的直接子元素。注意获取的是直接子元素,如果子元素中包括子元素是不会被选中的。比如页面结构:

```
<div>
  <p class="testClass">测试 1<span>子元素 1</span></p>
</div>
<br />
<div class="testClass">
  <p class="testClass">测试 2<span>子元素 2</span></p>
</div>
```

使用 children()函数:

```
$("div").children().css("border-color", "#FF0000");
```

上面的函数向被选中的元素添加了红色边框,如图 5-4 所示。注意到两个 span 元素都没有被选中,因为 span 的父结点是 p 元素,p 元素的父结点是 div。虽然 span 也是 div 的子结点,但不是直接子结点。

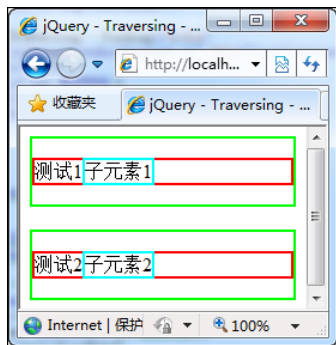


图 5-4 children()函数实例

#### 3. contents()函数

需要注意 contents()函数和 children()函数的区别。同样是上面的 HTML 结构,使用



children()函数:

```
$("div").children().children().wrap("<b/>");
```

的效果如图 5-5 所示, 文本结点没有被加粗:

对于 HTML 来说纯文字也是结点, 比如上例中的“测试 1”是 p 元素的文本结点, span 是 p 元素内的元素结点。第一次 children()函数选中了两个 p 元素, 再次使用 children()函数则选中 p 元素的直接子结点, “测试 1”和 span 元素都算是 p 元素的直接子结点, 但是 children()函数忽略了文本结点, 所以“测试 1”和“测试 2”这两个文本结点没有被加粗。

使用 contents()函数:

```
$("div").contents().contents().wrap("<b/>");
```

效果如图 5-6 所示, 文本结点也被加粗了。

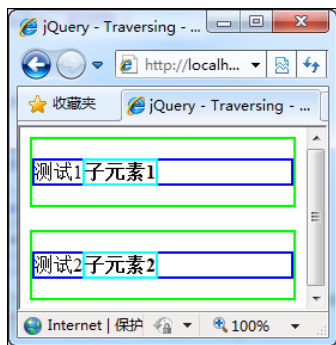


图 5-5 children()函数与 contents()函数的区别

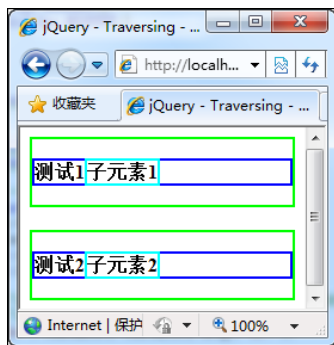


图 5-6 contents()函数效果

第一次使用 contents()函数会选中两个 p 元素, 再次调用 contents()函数会选中 p 元素的两个直接子元素“测试 1”和 span 结点。能选中文本结点是 contents()函数和 children()函数的主要区别。

#### 4. find()函数

find()函数是最常使用的函数之一, 在 jQuery 选择器一节就讲过 find()函数, filter()函数和 find()函数的不同请参见上文 filter()函数的讲解。

在大多数情况下, 下面两个语句返回的结果是一样的。

```
$('#wrap').find('div.box');  
$('#wrap div.box');
```

但是当使用的选择器表达式是“过滤器”时, 就会产生不同的结果。比如:

```
$("div.box").find("p:first");
```

上面的语句返回的集合个数可能大于 1, 因为它会对\$("div.box")选择出来的集合做 each 操作, 在每个对象上执行 find()方法。如果改成:

```
$("div.box p:first");
```

则返回的集合个数永远不会大于 1, 如图 5-7 所示。



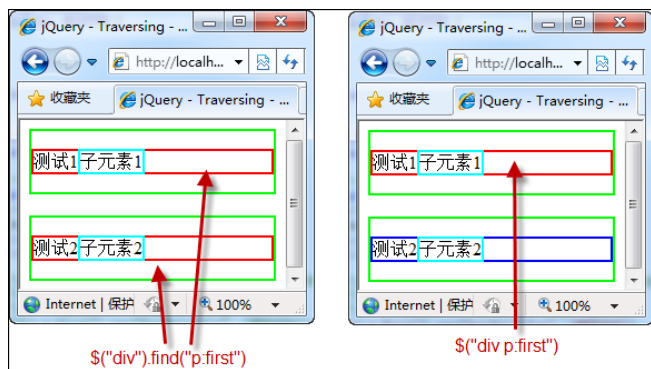


图 5-7 使用 find()方法和使用后代选择器的区别

## 5.4 串联函数——操作对象链

使用 Traversing 分类中的“过滤”和“查找”函数，会改变 jQuery 对象集合。串联函数的作用就是找回或合并这些变更。串联函数的清单如表 5-5 所示。

### 5.4.1 串联函数列表

表 5-5 串联函数列表

名 称	说 明	举 例
andSelf()	将先前所选的元素加入到当前元素中。 对于筛选或查找后的元素，要加入先前所选元素将会很有用	选取所有 div 及内部的 p，并加上 border 类： \$("div").find("p").andSelf().addClass("border");
end()	回到最近的一个“破坏性”操作之前，即将匹配的元素列表变为前一次的状态。 如果之前没有破坏性操作，则返回一个空集。所谓的“破坏性”就是指任何改变所匹配的 jQuery 元素的操作。这包括在 Traversing 中返回任何一个 jQuery 对象的函数-add, andSelf, children, filter, find, map, next, nextAll, not, parent, parents, prev, prevAll, siblings, slice 再加上 Manipulation 中的 clone, appendTo, prependTo, insertBefore, insertAfter, replaceAll	选取所有的 p 元素，查找并选取 span 子元素，然后再反过来选取 p 元素： \$("p").find("span").end()

### 5.4.2 串联函数要点

#### 1. andSelf()函数

andSelf()常和 Traversing 查找函数一起使用，用来获取两个操作合并后的对象集合。其经常和 find()一起使用，比如：

```
$("div").find("p").css("background-color", "#FFFF00");  
$("div").find("p").andSelf().css("border-color", "#FF0000");
```

效果如图 5-8 所示。

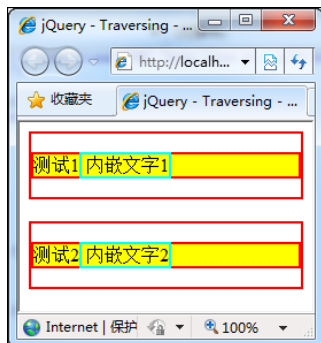


图 5-8 andSelf()函数实例

注意，当使用 find()函数后，jQuery 对象集合内的元素是两个 p 元素，将其背景色设置成了黄色。

在 find()函数后使用 andSelf()函数后，jQuery 对象集合有了变化，将 find()函数前一步的对象集合，即两个 div 元素和使用 find 后的 jQuery 对象集合，这两个 p 元素合并在了一起。所以这 4 个元素同时拥有了红色边框。

对于 Traversing 分类中的过滤函数，不应该使用 andSelf()函数，因为过滤出来的集合是前一步对象集合的子集。如果只是想获取过滤前的对象集合，应该使用下面介绍的 end()函数。

andSelf()函数是不能同时使用两次的，比如下面的语句中，第二次调用 andSelf()函数是没有效果的：

```
$("div").find("p").find("span").andSelf().andSelf().css("border-color", "#FF0000");
```

下面介绍的 end()函数是可以多次使用的。

## 2. end()函数

end()函数可以恢复一次破坏性操作。所谓的破坏性操作是指改变了 jQuery 对象集合中的元素的操作。主要有两大类：

Traversing 分类中的 add、andSelf、children、filter、find、map、next、nextAll、not、parent、parents、prev、prevAll、siblings、slice。

Manipulation 分类中的 clone、appendTo、prependTo、insertBefore、insertAfter、replaceAll。提供 end()函数主要是为了能够在 jQuery 链式操作上执行更多的操作。比如：

```
$("div").find("p").css("background-color", "#FFFF00").end().css("border-color", "#FF0000");
```

这一条 jQuery 链式命令，相当于执行了下面两句：

```
$("div").find("p").css("background-color", "#FFFF00");
$("div").css("border-color", "#FF0000");
```

end()函数使 jQuery 对象集合回到了执行 find()函数前的状态。

end()函数是可以多次使用的，比如：

```
$("div").find("p").find("span").end().end(); //相当于回到了$("div")时的状态
```





## 5.5 小结

本章讲解了如何应用 jQuery 中的 Traversing 分类函数管理 jQuery 对象集合，以及如何动态（on-the-fly）创建新的元素。学会了控制 jQuery 集合后，在接下来的章节中将学习如何操作集合元素，以及管理集合元素的属性和样式。





## 第 6 章



# 使用 jQuery 操作元素

在学习过 jQuery 的基础知识及如何管理 jQuery 对象集合后，本章将介绍如何使用 jQuery 操作元素，包括修改元素的属性和样式，在 jQuery 官方文档中分别对应操作属性的分类“Attributes”和操作 CSS 的函数分类“CSS”。

这两类函数都是最基础、最常用的，将这两类函数理解透彻，无论是对使用 jQuery 还是开发 jQuery 插件都大有裨益。



## 6.1 DOM 属性与 HTML 元素属性

在第 2 章中,已经简单介绍过 DOM 元素和 HTML 元素的区别。合格的 JavaScript 开发人员和页面设计人员应该时刻区分 DOM 元素和 HTML 元素、DOM 属性和 HTML 属性。

### 6.1.1 区分 DOM 属性与 HTML 元素属性

HTML 元素就是页面上的 HTML 代码,比如一个“img”标签:

```

```

其中的“src”、“id”、“alt”等就是 HTML 元素属性。

最终浏览器会解析 HTML,构建 DOM 模型,并最终展示给用户。也就是说 HTML 元素最后会被浏览器解析为 DOM 元素。可以使用 JavaScript 获取到 DOM 元素:

```
var myImg = document.getElementById("hibiscus");
```

在 JavaScript 中获取到的都是 DOM 元素,而不是 HTML 元素。可以获取 DOM 元素的属性,即 DOM 属性:

```
alert(myImg.alt);
```

因为 HTML 元素属性和 DOM 属性的名称及值大部分都相同,所以导致很多人错误地将两者认为是相同的,或统称为“属性”。但是 HTML 元素属性和 DOM 属性并不是完全一致的,比如“img”元素的 HTML 属性“src”的值是:

```
../../pic/image.1.jpg
```

这是一个相对路径,浏览器将其解析成 DOM 元素后,对应的 DOM 属性“src”的值被转换成了绝对路径:

```
http://www.jquerystrom.com/pic/image.1.jpg
```

使用 JavaScript 可以操作 DOM 属性,比如改变图片的 src 属性值:

```
myImg.src = "../../pic/image.2.jpg";
```

修改了 DOM 属性后,浏览器会重新解析 DOM,有的浏览器是重新解析并渲染全部 DOM 元素,有的是部分重新解析,无论是何种方式,最后都会将修改后的内容展示给用户。比如上面的代码修改了 img 元素的 src 属性,最后将图片“image2”展现给用户。

注意,页面的 HTML 是不会变化的,查看页面的 HTML 源代码,“myImg”的 HTML 代码并没有变化。但是在 FireFox 的 FireBug 或者 IE 8 的 IE Developer Tool 等页面调试工具的 HTML 窗口中,会显示这一变化,如图 6-1 和图 6-2 所示。



## 第 6 章 使用 jQuery 操作元素

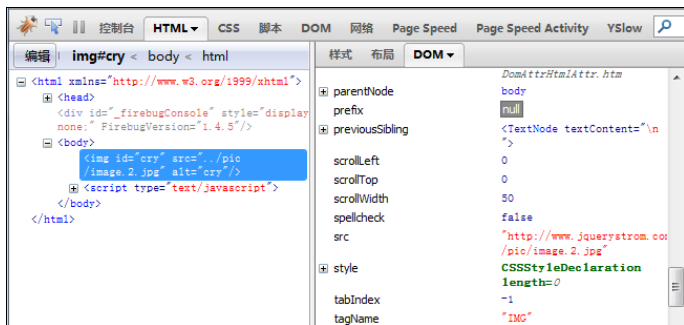


图 6-1 Firefox 中的 HTML 窗口

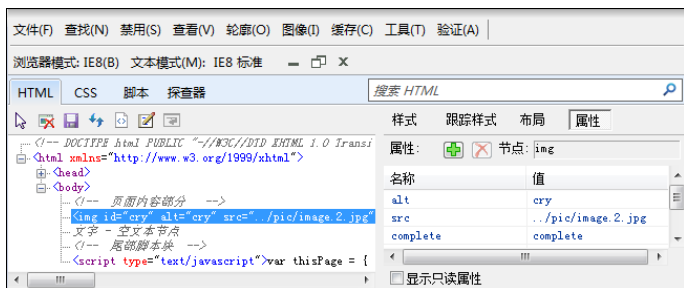


图 6-2 IE Developer Tool 中的 HTML 窗口

HTML 元素属性和 DOM 属性除了内容值不同外，有时属性名称也会有所不同，如图 6-3 所示为 CSS 样式类的 DOM 属性。比如元素的 CSS 样式类属性：

```

```

为 img 添加了 HTML 属性“class”，但是在 JavaScript 中无法使用“myImg.class”获取此属性值。这是因为 HTML 元素属性“class”对应的 DOM 属性是“className”：

```
alert(myImg.className);
```

区分 HTML 元素属性和 DOM 属性的名称差异曾经是一件考验经验和记忆力的事情。如今使用 jQuery 的属性操作函数 attr() 就可以“忘记”这些差异，下面将详细讲解。



图 6-3 CSS 样式类的 DOM 属性

### 6.1.2 使用 JavaScript 操作 DOM 属性

操作 DOM 属性不需要用到 jQuery，使用原生的 JavaScript 语言即可。使用 JavaScript 操作 DOM 属性就是操作 JavaScript 对象的属性：

```
var myImg = document.getElementById("cry"); //获取 ID 为 cry 的 DOM 对象
myImg.src = "../pic/image.2.jpg";          //修改 src 属性
```

JavaScript 对象的属性是不需要声明的，可以直接为未声明的属性赋值：

```
var myImg = document.getElementById("cry");
myImg.myProp = "自己的属性";
alert(myImg.myProp);
```

上例中 myImg 的 myProp 属性是没有声明过的。



除了使用“属性名”的形式（“.”运算符），还可以使用下面几种形式访问属性：

```
myImg.src = "../pic/image.2.jpg";    //使用“.”运算符
myImg["src"] = "../pic/image.2.jpg"; //使用属性访问器
var propName = "src";                //属性访问器支持变量
myImg[propName] = "../pic/image.2.jpg";
```

因为提供了属性访问器，所以可以通过下面的方式遍历一个 DOM 对象的所有属性：

```
//遍历 myImg 的属性
var result = "";
for (var p in myImg) {
    result += "属性名: " + p + ", 属性值: " + myImg[p] + "\n";
}
```

注意，事件或函数也是对象的一个属性。如果一个对象是 DOM 对象，则默认情况下就拥有很多的属性，图 6-4 是部分属性的截图。

注意最后一项 onkeydown，这是键盘按下事件，应该是一个函数。如果为 DOM 对象绑定了事件，则属性值就是函数的内容。比如在页面上为 img 元素添加事件：

```

```

则事件 onkeydown 被解析为 DOM 对象的 onkeydown 属性：

```
var myImg = document.getElementById("cry");
alert(myImg["onkeydown"]);
```

效果如图 6-5 所示。

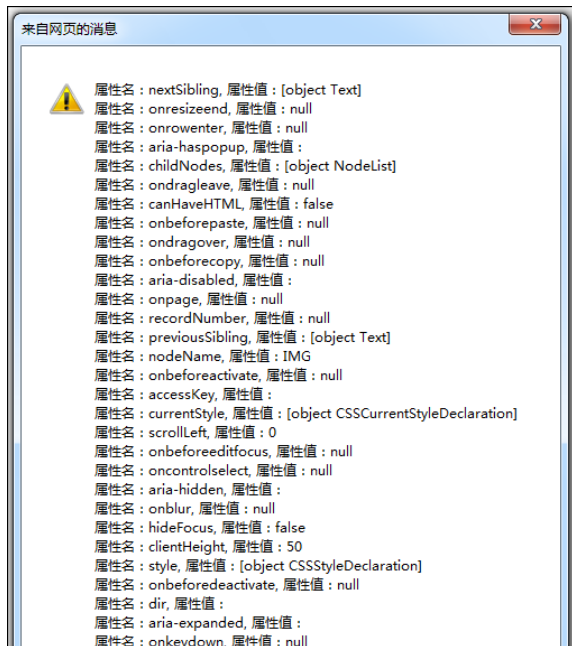


图 6-4 DOM 对象的默认属性



图 6-5 事件也是 DOM 元素的属性

### 6.1.3 使用 JavaScript 操作 HTML 元素属性

使用 JavaScript 中的 getAttribute 和 setAttribute，可以操作 HTML 元素属性。比如：



```
alert(myImg.getAttribute("class"));
myImg.setAttribute("class", "myClass2");
```

上面的代码会改变元素的样式类。通过改变 HTML 属性 “class”，会改变相应 DOM 元素的 “className” 属性。

使用 `getAttribute()` 和 `setAttribute()` 函数后，页面的 “HTML 源代码” 同样不会有变化，但是在 FireBug 和 IE Developer Tool 中的 HTML 内容窗口都有了变化。

虽然使用 `getAttribute` 和 `setAttribute` 可以操作 HTML 元素属性，但并不是所有的 HTML 元素属性都有对应的 DOM 属性。比如自定义的 HTML 元素属性就无法转换成 DOM 属性。再如元素属性 “className”：

```
myImg.setAttribute("className", "myClass2");
alert(myImg.className); // myClass
```

因为 DOM 属性 “className” 的值是和 HTML 元素属性 “class” 关联的，所以修改了 HTML 元素属性 “className” 后，仅仅是在 HTML 中能够观察到：

```

```

但是 DOM 属性 “className” 仍然为 “myClass”。

通常，将 DOM 对象理解为 JavaScript 对象，DOM 属性也就是 JavaScript 对象的属性。

## 6.2 使用 jQuery 操作 DOM

在 jQuery 中，提供了 “attributes” 函数分类用来操作 DOM 元素的属性和样式。

此分类的 jQuery 的官方 API 地址为 <http://api.jquery.com/category/attributes/>。

在新版的 jQuery API 文档中，分类方式发生了改变。有些函数会同时出现在多种分类中。比如 `attributes` 分类中的函数几乎都同时被包含在 `manipulate` 分类中。此分类包括下列函数，如表 6-1 所示。

表 6-1 attrbute 分类函数列表

函数名称	函数说明	所属分类
<code>.addClass</code>	为选中集中的元素添加样式	Attribute CSS Manipulate/Class Attribute
<code>.attr</code>	获取或设置元素的属性	Attribute Manipulate /General Attributes
<code>.hasClass</code>	判断选中的元素是否应用了指定的样式	Attribute CSS Manipulate/Class Attribute
<code>.html</code>	获取或设置元素中的 HTML 内容	Attribute Manipulate /DOM Insertion
<code>.removeAttr</code>	为匹配的对象集合移除属性	Attribute Manipulate /General Attributes



(续表)

函数名称	函数说明	所属分类
.removeClass	为匹配的对象集合移除样式	Attribute CSS Manipulate/Class Attribute
.text	为匹配的对象集合获取或设置文本内容	Attribute Manipulation/Dom Insertion Inside Manipulate /General Attributes
.toggleClass()	为匹配的对象集合添加或移除样式, 如果没有样式则添加, 有则移除	Attribute CSS Manipulate/Class Attribute
.val()	获取或设置匹配对象的表单值	Attribute Forms Manipulate /General Attributes

其中每个函数都有若干的重载方法。此分类中的函数按照功能, 分为操作元素属性、操作元素样式及改变 HTML 内容三类。

### 6.2.1 使用 jQuery 操作元素属性

下面是与操作元素属性相关的函数列表, 如表 6-2 所示。

表 6-2 attr()函数重载列表

函数签名	返回值	版 本	函数说明	参数说明
attr( attributeName )	String	1.0	取得第一个匹配元素的属性值。通过这个方法可以方便地从第一个匹配元素中获取一个属性的值。如果元素没有相应属性, 则返回 undefined	attributeName: 属性名称
.attr( attributeName, value )	jQuery	1.0	为所有匹配的元素设置一个属性值	attributeName: 属性名称 value: 属性值
.attr( map )	jQuery	1.0	将一个“名/值”形式的对象设置为所有匹配元素的属性	map: 含有名/值对的 JavaScript 对象
.attr( attributeName, function(index, attr) )	jQuery	1.1	为所有匹配的元素设置一个计算的属性值。 不提供值, 而是提供一个函数, 由这个函数计算的值作为属性值	attributeName: 属性名称 function(index, attr): 返回属性值的函数, 第一个参数为当前元素的索引值, 第二个参数为原先的属性值
.removeAttr( attributeName )	jQuery	1.0	从每一个匹配的元素中删除一个属性	attributeName: 属性名称

使用 attr()函数, 可以忽略 HTML 元素属性和 DOM 属性的区别。因为在 attr()的内部, 会首先寻找是否有名称为 attributeName 的 DOM 属性(使用上面介绍的 JavaScript 操作 DOM 属性的方法), 然后再判断是否有名称为 attributeName 的 HTML 属性(使用上面介绍的 JavaScript 操作 HTML 属性的方法)。并且 attr()函数不仅仅是简单地调用这两个方法, 其中有很多针对特殊属性、针对各种浏览器的处理。

下面是 jQuery 内部实现 attr() 函数的核心方法，对于大部分语句添加了中文注释：

```
attr: function(elem, name, value, pass)
{
    //判断结点类型， 如果没有传入结点，或者结点是 text 结点或注释结点，则返回 undefined
    if (!elem || elem.nodeType === 3 || elem.nodeType === 8) {
        return undefined;
    }
    if (pass && name in jQuery.attrFn) {
        return jQuery(elem)[name](value);
    }
    var notxml = elem.nodeType !== 1 || !jQuery.isXMLDoc(elem),
        set = value !== undefined; //是设置还是获取
    //格式化属性名称。如果在 jQuery 的 props 中则格式化属性名称，否则不变
    name = notxml && jQuery.props[name] || name;
    if (elem.nodeType === 1)
    {
        //如果是 HTML 结点，才进行下面的运算
        //是否需要特殊处理的属性，在 IE 下有以下属性需要特殊处理：href、src、style
        var special = rspecialurl.test(name);
        if (name === "selected" && !jQuery.support.optSelected) {
            var parent = elem.parentNode;
            if (parent) {
                parent.selectedIndex;
                if (parent.parentNode) {
                    parent.parentNode.selectedIndex;
                }
            }
        }
        if (name in elem && notxml && !special)
        {
            //使用 DOM 0 获取或设置 DOM 属性
            if (set) {
                //不允许 "type" 属性被修改，因为在 IE 下会引发问题
                if (name === "type" && rtype.test(elem.nodeName) && elem.parentNode) {
                    throw "type property can't be changed";
                }
                elem[name] = value;
            }
            //对于 form 元素的属性，使用 getAttributeNode 获取属性结点。1.4 中没有处理
            //“button” 元素，
            //button 元素应该使用同样的方式处理，否则在 IE 7 或更低版本 IE 中，将获取
            //不到 value 属性
            if (jQuery.nodeName(elem, "form") && elem.getAttributeNode(name)) {
                return elem.getAttributeNode(name).nodeValue;
            }
            if (name === "tabIndex") {
                var attributeNode = elem.getAttributeNode("tabIndex");
                return attributeNode && attributeNode.specified ?
                    attributeNode.value :
                    rfocussable.test(elem.nodeName) || rclickable.test(elem.nodeName) && elem.
href ?
                        0 :
                        undefined;
            }
            return elem[name];
        }
    }
    if (!jQuery.support.style && notxml && name === "style")
    {
        //对于 style 属性进行特殊处理。通过设置 cssText 可以使用 JavaScript 修改元素样式
        if (set) {
            elem.style.cssText = "" + value;
        }
        return elem.style.cssText;
    }
}
```



```

    if (set)
    {
        //如果没有相应的 DOM 属性，则尝试使用 setAttribute 和 getAttribute 修改 HTML 属性
        //"" + value 这种表达式会将 value 转换为 string 类型，否则 IE 下可能会出现错误
        elem.setAttribute(name, "" + value);
    }
    var attr = !jQuery.support.hrefNormalized && notxml && special ?
    //在 IE 下有一些属性需要特殊处理：href、src、style
    //hrefNormalized: 如果浏览器从 getAttribute("href")返回的是原封不动的结果，则返回 true。在 IE 7（含）及以下版本中会返回 false，因为它的 URL 已经被转换成了绝对地址。
        elem.getAttribute(name, 2) :
        elem.getAttribute(name);
    return attr === null ? undefined : attr;           //如果属性不存在则返回 undefined
}
//设置 style 属性，不赞成使用 attr 设置 style 属性。请使用 style 替换
return jQuery.style(elem, name, value);
}
});

```

如果想彻底了解 attr()函数究竟做了哪些事情，就需要仔细地阅读上面的代码，从中会发现很多的知识点。有了 attr()函数，可以忽略 HTML 元素属性与 DOM 属性的区别了，因为都可以通过 attr()函数获取或设置。attr()函数具有以下优点：

- ❑ 统一了 HTML 属性与 DOM 属性，比如统一了 class 和 className 的名称，同时支持 class 和 className 设置元素的样式。
- ❑ 统一了浏览器差异，对特殊属性做了处理。比如 src 属性等。保证在所有浏览器中返回相同的结果。
- ❑ 设置属性值后返回 jQuery 对象，可以实现 jQuery 链式操作。

下面的例子演示 attr()函数格式化 class 和 className 这个特殊的元素属性。

```

//HTML: <div id="divMsg" class="myClass"></div>
alert($("#divMsg").attr("class")); //输出: myClass
alert($("#divMsg").attr("className")); //输出: myClass

```

下面的例子演示 attr()函数对于 src 属性始终返回 HTML 元素属性：

```

//HTML: 
//IE 7 即更低版本 IE 中返回"http://localhost/pic/image.1.jpg"
//FF 或 IE 8 返回 "../pic/image.1.jpg"
alert($("#img1")[0].getAttribute("src"));
alert($("#img1")[0].src);           //始终返回 "http://localhost/pic/image.1.jpg"
alert($("#img1").attr("src")); //始终返回 "../pic/image.1.jpg"

```

使用 attr()函数获取属性值时，需要注意以下事项：

- ❑ 只获取匹配元素集合的第一个元素的属性值，如果属性值不存在则返回 undefined。
- ❑ 对于 src、href 属性始终返回 HTML 元素属性的值（非转换后的 DOM 属性）。

理解了前面介绍的 HTML 属性和 DOM 属性的区别后，对于这两个值的不同将不再有任何疑惑。使用 attr()函数获取 HTML 属性值时需要注意，此方法只获取匹配元素中第一个元素的属性值。如果需要获取匹配元素的所有成员的属性值，则需要借助 each()方法遍历元素集合：

```

var array = new Array();
$("#img").each(function(i) {
    array.push($(this).attr("src"));
});

```

```
});  
alert(array.join("\n"));
```

attr()函数有三个用于设置属性的重载方法。假设有 HTML 片段：

```

```

设置单个属性：

```
$('#myImg').attr('alt', 'changed 1');
```

可以使用 attr(map)方法一次设置多个属性值：

```
$('#myImg').attr({  
    alt: 'changed 2',  
    title: 'myTitle 2'  
});
```

用这种形式修改属性值时，属性的名称可以添加引号：

```
$('#myImg').attr({  
    "alt": 'changed 3',  
    "title": 'myTitle 3'  
});
```

### 注意

如果是设置 class 属性，则必须添加引号。

在 1.4 版本中，添加了.attr( attributeName, function(index, attr) )函数重载，属性值可以是一个返回属性值的函数：

```
$('#myImg').attr("title", function(index, attr) {  
    return this.alt + "-" + index + "-" + attr  
});
```

函数的返回值将作为属性值被设置，函数的两个参数 index 表示元素在当前集合中的索引值，attr 表示旧的属性值，函数中的 this 是 DOM 对象。

使用 attr()函数设置属性值时，如果属性已经存在则执行修改操作，如果属性值没有则执行添加操作。

如果想要删除某一个属性，可以使用 removeAttr()函数：

```
//HTML:<input type="text" disabled="disabled" value="myValue" />  
var input = $('input')[0];  
alert("disabled:" + input.disabled + ", value:" + input.value); //输出: "disabled:true, value:myValue"  
$(input).removeAttr("disabled");  
$(input).removeAttr("value");  
alert("disabled:" + input.disabled + ", value:" + input.value); //输出: "disabled:false, value:"
```

上面这个例子中，使用 removeAttr()函数移除“value”属性后，value 值被清空。但是移除了“disabled”属性后，“disabled”属性值没有被清空，而是设置为了默认值“false”。也就是说使用 removeAttr()函数可以清空元素的 HTML 属性，但是对于必须有默认值的属性，删除属性后会将此属性设置为默认值，而不是清空。上例中的 disabled 和 value 属性都是被恢复成了默认值“false”和空字符串。如果是自定义属性，则可以立刻被删除，删除后此属性的值变为“undefined”：



```
input.myAttr = "myAttr";
alert(input.myAttr);    // 输出: myAttr
$(input).removeAttr("myAttr");
alert(input.myAttr);    //输出: undefined
```

DOM 属性和 HTML 属性是相互映射的,上面的例子说明添加了自定义的 DOM 属性后,也会为元素添加 HTML 属性,并且同样可以通过 `removeAttr()` 函数删除属性。

## 6.2.2 使用 jQuery 操作元素 CSS

部分控制元素 CSS 的方法被放在了 `attribute` 分类中,但是并不是全部。jQuery 单独设置了 CSS 分类用来归类所有和 CSS 操作有关的方法,如表 6-3 所示。

表 6-3 CSS 分类函数列表

函数名称	函数说明	所属分类
<code>.addClass()</code>	为选中集中的元素添加样式	Attribute CSS Manipulate/Class Attribute
<code>.css()</code>	获取或设置元素的属性	CSS Manipulate/Style Properties
<code>.hasClass()</code>	判断选中的元素是否应用了指定的样式	Attribute CSS Manipulate/Class Attribute
<code>.removeClass()</code>	从所有匹配的元素中删除全部或者指定的类	Attribute CSS Manipulate/Class Attribute
<code>.toggleClass()</code>	如果存在(不存在)就删除(添加)一个类	Attribute CSS Manipulate/Class Attribute
<code>.height()</code>	设置或获取元素当前计算的高度值	CSS Dimensions Manipulate/Style Properties
<code>.width()</code>	设置或获取元素当前计算的宽度值	CSS Dimensions Manipulate/Style Properties
<code>.innerHeight()</code>	设置或获取元素内部区域高度(包括补白、不包括边框)	CSS Dimensions Manipulate/Style Properties
<code>.innerWidth()</code>	设置或获取元素内部区域宽度(包括补白、不包括边框)	CSS Dimensions Manipulate/Style Properties
<code>.outerHeight()</code>	设置或获取元素外部区域高度(默认包括补白和边框)	CSS Dimensions Manipulate/Style Properties



(续表)

函数名称	函数说明	所属分类
.outerWeight()	设置或获取元素外部区域宽度（默认包括补白和边框）	CSS Dimensions Manipulate/Style Properties
.position()	获取匹配元素相对父元素的偏移	CSS Offset Manipulate/ Style Properties
.offset()	获取匹配元素在当前视口的相对偏移	CSS Offset Manipulate/ Style Properties
.scrollLeft()	设置或获取匹配元素相对滚动条左侧的偏移	CSS Offset Manipulate/ Style Properties
.scrollTop()	设置或获取匹配元素相对滚动条顶部的偏移	CSS Offset Manipulate/ Style Properties

在 1.4 版本中，函数分类有了较大的变化，尤其是多了一级分类“offset”和“Dimensions”。这两个分类中的所有函数又都被包含在“CSS”分类中。也就是说 CSS 中的函数可以分为操作 CSS、位移、测量三部分。首先来看和 CSS 相关的函数，如表 6-4 所示。

表 6-4 CSS 函数重载列表

函数签名	返回值	版 本	函数说明	参数说明
.addClass( classNames )	jQuery	1.0	为每个匹配的元素添加指定的样式类	classNames: 一个或多个要删除的 CSS 类名，用空格分开
.addClass( function(index, class) )	jQuery	1.4	为每个匹配的元素添加 function 返回的样式类	function(index, class): 返回一个或多个空格分隔的 class 名。 index 参数为对象在这个集合中的索引值，class 参数为这个对象原先的 class 属性值
.removeClass( [ classNames ] )	jQuery	1.0	从所有匹配的元素中删除全部或者指定的类	classNames: 一个或多个要删除的 CSS 类名，用空格分开
.removeClass( function(index, class) )	jQuery	1.4	从所有匹配的元素中删除 function 返回的样式类	function(index, class): 返回一个或多个空格分隔的 class 名。 index 参数为对象在这个集合中的索引值，class 参数为这个对象原先的 class 属性值
.css( propertyName )	String	1.0	获取匹配集合第一个元素的指定 CSS 属性的值	propertyName: CSS 属性名
.css( propertyName, value )	jQuery	1.0	为匹配集合的所有元素设置 CSS 属性	propertyName: CSS 属性名 value: 要设置的值





(续表)

函数签名	返回值	版 本	函数说明	参数说明
<code>.css( propertyName, function(index, value) )</code>	jQuery	1.4	为匹配集合的所有元素设置 CSS 属性, 值为函数的返回值	propertyName: CSS 属性名 function(index, value): 返回要设置的 CSS 属性值。index 参数为对象在这个集合中的索引值, value 参数为原始的 CSS 属性值
<code>.css( map )</code>	jQuery	1.0	把一个“名/值对”对象设置为所有匹配元素的样式属性。这是一种在所有匹配的元素上设置大量样式属性的最佳方式	map: 含有名/值对的 JavaScript 对象
<code>toggleClass( classNames )</code>	jQuery	1.0	如果存在(不存在)就删除(添加)样式类	classNames: 一个或多个要删除的 CSS 类名, 用空格分开
<code>.toggleClass( classNames, switch )</code>	jQuery	1.3	如果存在(不存在)就删除(添加)样式类	c classNames: 一个或多个要删除的 CSS 类名, 用空格分开。 switch: 布尔值, 决定元素是否包含样式类
<code>.toggleClass( function(index, class), [ switch ] )</code>	jQuery	1.4	如果存在(不存在)就删除(添加)样式类。 如果开关 switch 参数为 true 则加上对应的 class, 否则就删除	function(index, class): 返回一个或多个空格分隔的 class 名。 index 参数为对象在这个集合中的索引值, class 参数为这个对象原先的 class 属性值

控制 CSS 样式类的函数主要有三个: `addClass()`、`removeClass()`和 `toggleClass()`, 分别用于添加、移除和切换 CSS 样式类。

这三个函数的用法十分相同, 唯一的不同就是 `removeClass()`函数可以不传递参数, 表示移除所有的 class 类。当传递 CSS 样式类时, 三个函数都可以同时传递多个用空格隔开的 CSS 样式类。

```
//HTML:<div id="divMsg">测试文字</div>
this.$divMsg = $("#divMsg");
this.$divMsg.addClass("bred tblue");
alert("class has been added!");
this.$divMsg.removeClass("bred tblue");
alert("class has been removed!");
this.$divMsg.toggleClass("bred tblue");
alert("class has been toggled!");
this.$divMsg.toggleClass("bred tblue");
```

在最新的 1.4.1 版本中, 不会对元素添加已经存在的样式类, 即使用 `addClass` 不会出现 `class="foo bar foo"`这种情况。在 `addClass` 的内部, 会检测元素是否已经应用了要添加的样式类, 如果存在则不添加。

在 1.4 版本中允许使用 `function` 计算样式类, `addClass()`、`removeClass()`和 `toggleClass()`三个函数的用法相同, 下面以 `toggleClass()`函数为例说明 `function` 的用法。

```
this.$divMsg.toggleClass(function(index,class){
    alert(class);
    return "bred tblue";
```



```
});
alert("class has been toggled!");
this.$divMsg.toggleClass(function(index,class){
    alert(class);
    return "bred tblue";
});
```

可以将需要替换的 CSS 样式类用函数的形式替换，函数的签名是 `function(index,class)`，`index` 是元素的集合的索引值，`class` 是元素原始的 CSS 样式类字符串，用空格分隔。`function` 需要返回一个或多个用空格分隔的 CSS 样式类字符串。`addClass()` 和 `removeClass()` 函数都提供了同样功能。

`toggleClass()` 函数可以传递一个可选的 `boolean` 类型的参数 “switch”，这是一个强制开关，`true` 表示总是添加样式，`false` 表示总是删除样式。这个功能和 `addClass` 和 `removeClass` 有些重复，导致使用 `toggleClass()` 完全可以替代 `addClass()` 和 `removeClass()` 这两个函数。但是推荐尽量使用 `addClass()` 和 `removeClass()`，因为这两个方法更利于理解和阅读。

如果想移除所有的 CSS 样式类并添加新的样式类，可以用两种方式：

```
this.$divMsg.removeClass().addClass("newClass");
this.$divMsg.attr("class", "newClass");
```

虽然从效率上讲 `attr()` 函数略高，但是可以忽略，并且 `removeClass()` 和 `addClass()` 拥有更好的可读性。

大部分代码中应该使用 CSS 样式类包装样式，尽量避免在 HTML 代码中出现内联样式：

```
<div id="divMsg" style=" border: solid 1px #FF0000; ">测试文字</div>
```

样式应该和程序代码一样，需要进行封装和复用。

但是 jQuery 也提供了单独处理 CSS 样式的方法：`css()`，用于获取或修改 CSS 样式。

使用 `css(propertyName)` 函数重载可以获取某一个样式属性的值。使用此函数的好处是可以实现多浏览器统一，比如在 IE 中 `float` 属性的名称是 “`styleFloat`”，而在 W3C 标准的浏览器中是 “`cssFloat`”，使用 jQuery 的 `css()` 函数就可以统一这些变化：

```
$(div.left').css('float');
$(div.left').css('cssFloat');
$(div.left').css('styleFloat');
```

上面的三个函数都可以获取 `float` 的值（比如 “`left`”）。

并且 `css()` 函数支持省略连字符 “-” 的样式属性名，比如下面两个方法返回同样的结果：

```
$(div.left').css("background-color");
$(div.left').css("backgroundColor");
```

需要注意的是，对于缩略的 CSS 属性，比如 “`border`”，“`background`” 等，`css()` 方法不支持。其实虽然使用 `css()` 方法有时也能返回结果，比如 `css("border")`，但是由于没有进行特殊的处理，在不同浏览器中返回的结果是不同的。在 IE 8 中返回的颜色是 “`#ff0000`”，而 FireFox 中返回的是 “`rgb(255,0,0)`”。再比如 `css("margin")`，如果一个 `div` 元素没有设置过 `margin` 属性，则在 IE 8 中返回 “`auto`”，而在 FireFox 中返回空字符串。

所以如果不希望特殊处理这种缩略的 CSS 属性在不同浏览器的差异，请使用完整的 CSS 属性名替代。比如如果希望获取 `margin` 属性，则可以分别获取四个 `margin` 子属性。

```
alert(this.$divMsg.css("margin-top"));
```





```
alert(this.$divMsg.css("margin-right"));
alert(this.$divMsg.css("margin-bottom"));
alert(this.$divMsg.css("margin-left"));
```

目前在 1.4.1 版本中, 仍然存在着一些问题, 对于没有设置 margin 的元素, 在 IE 8 中返回 “auto”, 而 FireFox 中返回 “0px”。所以在获取没有设置的 CSS 属性前需要特别小心浏览器默认样式值的不同。

和 attr() 函数一样, css() 函数支持下面三种方式设置 CSS 属性值:

```
this.$divMsg.css("margin-left", "10px");
this.$divMsg.css("margin-left", function(index, value) { return "10px"; });
this.$divMsg.css({"margin-left": "10px", "margin-top": "10px"});
```

可以一次设置一个 CSS 属性, 使用 function 设置 CSS 属性或者一次设置多个 CSS 属性。

同样要注意, 当使用 .css(map) 传递一个名值对的对象, 一次设置多个 CSS 属性时, map 对象的属性名如果有特殊的字符, 应该添加引号。比如针对 “margin-left” 这个属性, 如果改成下面的语句就会出错:

```
this.$divMsg.css({margin-left : "10px", "margin-top": "10px"});
```

这是因为连字符 “-” 属于属性名中的特殊字符。所以总是在所有的属性名称上使用引号是一个好的习惯, 会避免这类错误的发生。

上面介绍了 CSS 分类中和 CSS 样式类及 CSS 属性相关的函数。注意到在 CSS 分类中还有很多的函数没有讲到, 这就是 1.4 新增加的分类 “Offset” 和 “Dimensions”。下面将这两个分类进行单独的讲解。

### 6.2.3 偏移量 offset 分类函数

Offset 是 jQuery 1.4 中新增加的分类, 里面的函数都同时包括在 CSS 分类下。

此分类下的函数都是和计算元素位置有关的。下面的函数列表, 如表 6-5 所示。

表 6-5 Offset 分类函数重载列表

函数签名	返回值	版 本	函数说明	参数说明
.offset()	Object, 拥有两个属性 top 和 left	1.2	获取第一个匹配元素相对于 document 对象的坐标。此方法只对可见元素有效	
.offset( coordinates )	jQuery	1.4	设置匹配元素相对于 document 对象的坐标。 .offset() 方法可以让我们重新设置元素的位置。这个元素的位置是相对于 document 对象的。如果对象原先的 position 样式属性是 static, 则会被改成 relative 实现重定位	coordinates: 一个对象, 必须包含 top 和 left 属性且属性值为 int 类型
.offset( function(index, coords) )	jQuery	1.4	设置匹配元素相对于 document 对象的坐标	function(index, coords): 返回一个对象, 必须包含 top 和 left 属性且属性值为 int 类型。 index 参数是元素的索引, coords 参数是当前的坐标对象



(续表)

函数签名	返回值	版 本	函数说明	参数说明
.position()	Object, 拥有两个属性 top 和 left	1.2	获取第一个匹配元素相对父元素的偏移坐标。 返回的对象包含两个整型属性: top 和 left。为精确计算结果, 请在补白、边框和填充属性上使用像素单位。此方法只对可见元素有效	
.scrollLeft()	int	1.2.6	获取匹配元素相对滚动条顶部的偏移。此方法对可见和隐藏元素均有效	
.scrollLeft( value )	jQuery	1.2.6	传递参数值时, 设置垂直滚动条顶部偏移为该值。 此方法对可见和隐藏元素均有效	value: int 类型
.scrollTop()	int	1.2.6	获取匹配元素相对滚动条左侧的偏移。此方法对可见和隐藏元素均有效	
.scrollTop( value )	jQuery	1.2.6	传递参数值时, 设置水平滚动条左侧偏移为该值。 此方法对可见和隐藏元素均有效	value: int 类型

.offset()方法和.position()方法都可以获取元素的坐标, 不同之处在于.offset()方法获取到的是相对于 document 的偏移坐标, 也就是相对于页面左上角的起点的偏移量, 而.position()获取到的是相对于第一个 position 属性设置为 relative 的父类的偏移量。

下面的例子演示如何使用 Offset()函数获取元素的偏移量:

```
var p = $("p:last");
var offset = p.offset();
p.html( "left: " + offset.left + ", top: " + offset.top );</script>
```

效果如图 6-6 所示。

获取到的偏移量, 常常用于确定页面弹出层的位置。在一个页面通过弹出层可以实现不跳转和刷新页面的情况下, 完成复杂的用户交互, 是增强用户体验的常用手段。

下面的示例演示制作弹出层的最基础的方法:

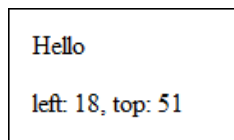


图 6-6 Offset 获取坐标

【代码路径: jQueryStorm.Web/ chapter6/ Demo-5-Offset.htm】

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery - Offset 函数实例</title>
  <script src="../../Static/common/js/jquery-1.4.2.js" type="text/javascript"></script>
</head>
<body>
  <br /><br /><br /><br /><br />
  <button id="btn" style="margin-left:40px;">弹出按钮</button>
  <br /><br /><br /><br /><br />
  <div id="divMsg" style="border: solid 1px; position:absolute; display:none;">测试文字</div>
  <script type="text/javascript">
    $(function ()
    {
      $("#divMsg").hide();
      $("#btnShow").click(function (e)
```



```

    {
        var offset = $(e.target).offset();           //获取按钮的 top 和 left 值
        offset.top += $(e.target).outerHeight();      //加上按钮元素的高度
        $("#divMsg").css(offset);                    //设置弹出层的位置
        $("#divMsg").toggle();                       //切换弹出层的显示状态
    });
    });
</script>
</body>
</html>

```

通过 `this` 指针获取到了触发 `click` 事件的对象，使用 `Offset()` 函数计算事件对象在页面上的坐标。

```
var offset = $(e.target).offset();           //获取按钮的 top 和 left 值
```

`outerHeight()` 函数，是用来获取元素包括补白和边框的外部高度，否则弹出层的位置会和按钮重叠，此部分将在本章后面进行讲解。`toggle()` 函数用来切换弹出层的显示状态。

最后使用 `CSS()` 函数重新设置了弹出层的 `top` 和 `left` 值。

在 jQuery 1.4 版本中，`Offset()` 函数还可以用来设置元素的 `top` 和 `left` 值，但是不支持对隐藏元素的设置，在本示例中因为元素的显示状态是变化的。如果在消失时使用 `Offset()` 函数设置元素的 `top` 和 `left`，则会产生位置计算不准确的错误。使用 `CSS()` 函数就不会有问题。本实例的效果如图 6-7 所示。

`position()` 函数的用法和 `Offset()` 函数相同，`position()` 函数只有一个无参数的重载方法，用户获取元素的相对于第一个 `position` 为 `relative` 父类元素的偏移量，此偏移量实际上就是 `CSS` 属性的 `top` 和 `left` 的值。

目前最新的 1.4.2 版本中，`position()` 获取到的 `left` 在 Chrome 浏览器存在着计算错误的 bug，预计会在后续版本中修复。

`scrollLeft()` 和 `scrollTop()` 用来获取或设置元素水平或垂直的滚动条举例。如果元素没有滚动条，则获取到的值为 0，设置滚动举例也不会有任何效果。比如下面的语句可以让滚动条向左滚动 300 像素。

```
$("#div.demo").scrollLeft(300);
```

效果如图 6-8 所示。

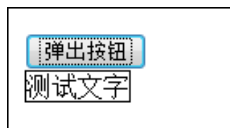


图 6-7 使用 `Offset()` 函数计算弹出层坐标

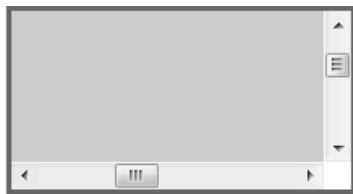


图 6-8 使用 `scrollLeft()` 函数

## 6.2.4 用于测量的 Dimensions 分类函数

`Dimensions` 分类保存了高度和宽度相关的函数。下面是 `Dimensions` 分类中的所有函数重载列表，如表 6-6 所示。

表 6-6 Dimensions 分类函数重载列表

函数签名	返回值	版 本	函数说明	参数说明
.height()	int	1.0	获取第一个匹配元素计算后的高度 (px)	
.height( value )	jQuery	1.0	设置元素的高度。 如果没有明确指定单位 (如: em 或 %), 则使用 px	value: 要设置的高度值。如果传递 int 类型则表示单位是 px, 否则请传递带有单位的字符串, 比如 1em
.height( function(index, height) )	jQuery	1.4	设置元素的高度, 高度值通过函数返回。 如果没有明确指定单位 (如: em 或 %), 则使用 px	function(index, height): 返回要设置的数值。index 参数是元素在原先集合中的索引位置, height 参数为原先的高度
.innerHeight()	int	1.2.6	获取第一个匹配元素内部区域高度 (包括补白、不包括边框)。 此方法对可见和隐藏元素均有效	
.innerWidth()	int	1.2.6	获取第一个匹配元素内部区域宽度 (包括补白、不包括边框)。 此方法对可见和隐藏元素均有效	
.outerHeight([ includeMargin ])	int	1.2.6	获取第一个匹配元素外部高度 (默认包括补白和边框)。 此方法对可见和隐藏元素均有效	includeMargin: 是否包括 Margin, 默认为 false
.outerWidth([ includeMargin ])	int	1.2.6	获取第一个匹配元素外部宽度 (默认包括补白和边框)。 此方法对可见和隐藏元素均有效	includeMargin: 是否包括 Margin, 默认为 false
.width()	int	1.0	取得第一个匹配元素当前计算的宽度值 (px)。可以用来获取 window 和 document 的宽	
.width( value )	jQuery	1.0	为每个匹配的元素设置 CSS 宽度 (width) 属性的值	value: 要设置的宽度值。如果传递 int 类型则表示单位是 px, 否则请传递带有单位的字符串, 比如 1em
.width( function(index, width) )	jQuery	1.4.1	为每个匹配的元素设置 CSS 宽度 (width) 属性的值	function(index, width): 返回要设置的数值。index 参数是元素在原先集合中的索引位置, width 参数为原先的宽度

从列表中可以看出, 元素的宽度和高度分为计算值 (height 和 width)、内部值 (innerHeight 和 innerWidth) 和外部值 (outerHeight 和 outerWidth)。

通过图 6-9、图 6-10 和图 6-11 可以清楚地看出它们的区别:

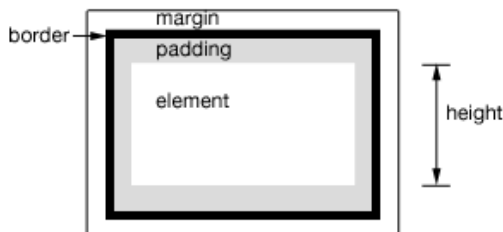


图 6-9 宽度和高度计算值



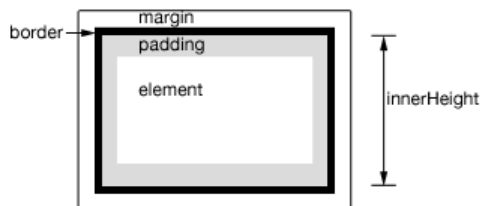


图 6-10 宽度和高度内部值

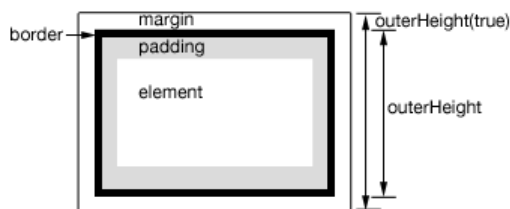


图 6-11 宽度和高度外部值

不带参数的函数获取到的是 `int` 数字类型返回值(单位是 `px`)。也可以使用 `CSS(“height”)` 和 `css(“width”)` 获取带有单位的高度和宽度的值, 比如: `100px`。显然使用 `int` 类型更易于计算。

设置函数的参数支持 `int` 和 `string`, 传递 `int` 表示单位为 `px`, 否则可以传递带有单位长度的字符串, 比如 `100pt`。

同样参数值可以使用 `function()` 函数替代, 下面是使用 `height()` 函数设置高度的示例代码:

```
$divMsg = $("#divMsg");
$divMsg .height(100);
$divMsg.height("150pt");
$divMsg.height(function(index, height) { return 300; });
```

## 6.2.5 使用 jQuery 改变元素内容

除了改变元素属性, 常常还要改变元素的内容。jQuery 提供了 `.html()` 和 `.text()` 函数用来获取和设置元素的内部 HTML 或文字, `html()` 与 `text()` 函数重载列表如表 6-7 所示。

表 6-7 `html()` 与 `text()` 函数重载列表

函数签名	返回值	版本	函数说明	参数说明
<code>.html()</code>	String	1.0	获取第一个匹配元素的 HTML 内容	
<code>.html( htmlString )</code>	jQuery	1.0	设置所有匹配元素的 HTML 内容	htmlString: HTML 字符串
<code>.html( function(index, html) )</code>	jQuery	1.4	设置所有匹配元素的 HTML 内容, HTML 值由函数返回	function(index, html): 返回要设置的 HTML 字符串。index 参数是元素在原先集合中的索引位置, html 参数为原先的 HTML 内容
<code>.text()</code>	String	1.0	获取第一个匹配元素的文字内容	
<code>.text( textString )</code>	jQuery	1.0	设置所有匹配元素的文字内容	textString: 文本内容字符串
<code>.text( function(index, text) )</code>	jQuery	1.4	设置所有匹配元素的文字内容, 文字值由函数返回	function(index, text): 返回要设置的文本串。index 参数是元素在原先集合中的索引位置, text 参数为原先的文本内容

.html()和.text()用来改变元素的内容，是经常要使用的方法。当用于获取元素的内容时，.html()返回元素内部连同 HTML 标签的内容，而.text()只返回文字部分。下面的例子可以很好地区别这两个函数。

【代码路径：jQueryStorm.Web/chapter6/Demo-7-HtmlText.htm】

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery - html() text()函数实例</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
</head>
<body>
  <!-- 页面内容部分 -->
  <div id="div1" style="border: solid 1px;"><span>图层 1</span></div>
  <br />
  <div id="div2" style="border: solid 1px;"><span>图层 2</span></div>
  <!-- 尾部脚本块 -->
  <script type="text/javascript">
    $(function() {
      alert($("#div1").html()); //输出为: "<span>图层 1</span>"
      alert($("#div2").text()); //输出为: "图层 2"
      $("#div1").html("<div>123</div>");
      $("#div2").text("<div>123</div>");
    });
  </script>
</body>
</html>
```

获取内容的效果如图 6-12 和图 6-13 所示。

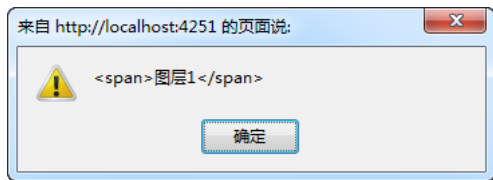


图 6-12 使用.html()函数获取内容

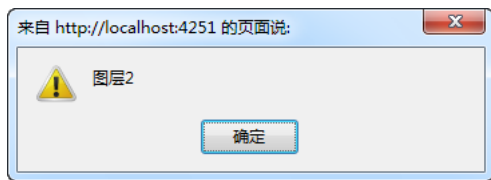


图 6-13 使用.text()函数获取内容

设置内容的效果如图 6-14 所示。

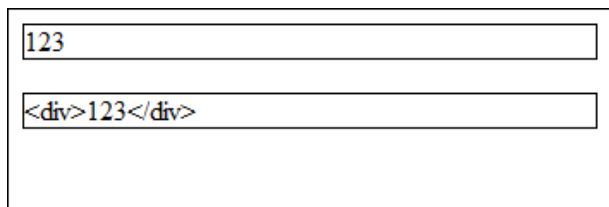


图 6-14 使用.html()和.text()函数设置元素内容

从实例可以看出，使用.html()返回的内容带有 HTML 标签，.html()无论是设置还是获取，内部都是使用“innerHTML”方法实现的。而使用.text()获取内容时，jQuery 使用了递归算法，并且使用 DOM 的“nodeValue”属性获取文本结点的值。使用.text()设置内容时，使用“createTextNode()”方法创建文本结点，所以如果文本结点是一个 HTML 字符串，其中的 HTML 特殊字符都会进行 HTML 编码。而.html()方法则会插入原始的 HTML 内容。

在 1.4 版本中，和大多数设置函数一样，.html()和.text()函数都提供了 function 设置值的







重载函数:

```
$("#div1").html(function(index, html) { return "<div>" + index + "</div>"; });  
$("#div2").text(function(index, text) { return "<div>" + index + "</div>"; });
```

## 6.3 小结

本章主要讲解了如何控制元素，首先要明确地区分 HTML 属性和 DOM 属性的区别和联系，并且介绍了如何使用原生的 JavaScript 语言修改元素的 HTML 属性和 DOM 属性。这些知识都作为进一步理解 jQuery 的 attr() 函数的内部实现。

在讲解使用 jQuery 操作元素时，分为三个部分，分别讲解如何控制元素属性、如何控制元素样式以及如何控制元素内容。其中控制元素样式部分又分成了三部分分别讲解 CSS 分类、控制坐标的 Offset 分类和控制宽高的 Dimensions 分类，1.4 版本的 jQuery 分类相比以往更加清晰，但是同时改动也比较大，使用以前版本的开发人员需要花一些时间去适应。

通过本章的学习，开发人员已经可以完全地控制页面元素了。在下面的章节将介绍页面开发中最重要的事件。







## 第 7 章



# 事件与事件对象

事件是脚本编程的灵魂，程序与用户的交互都是通过事件完成的，比如按钮单击事件。程序自身也有很多的时间，比如页面加载事件。本章将从基础开始讲解，介绍基础的事件模型和事件处理机制，以及如何使用 jQuery 使开发人员从事件噩梦中解放出来。

## 7.1 DOM 事件模型

DOM 事件模型是 W3C 定义的标准事件模型,因为 DOM 的标准分为不同的级别:level0、level1、level2、level3。

而且目前各个浏览器实现的 DOM 标准级别各不相同,所以很难在所有浏览器中通用 DOM 事件模型。但是作为合格的开发人员,要懂得什么是标准的事件模型。在本节开头讲解 DOM 事件模型,并紧接着讲解 jQuery 事件模型——一种可以跨浏览器的事件模型解决方案。

### 7.1.1 DOM 事件流

DOM 事件流分为两个阶段,即事件捕获阶段和事件冒泡阶段,如图 7-1 所示。当一个事件被触发,首先从根结点开始捕获事件,一直捕获到叶子结点。然后开始从叶子结点执行事件冒泡,冒泡到根结点即完成整个事件流。此过程如图 7-1 所示。

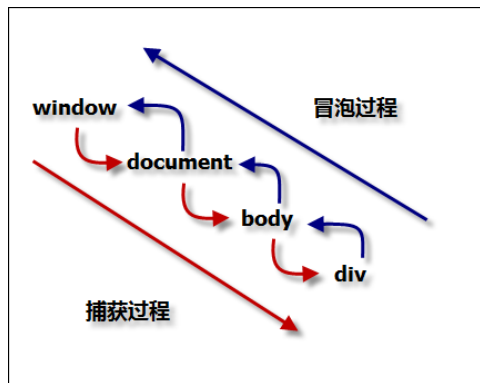


图 7-1 DOM 事件流

在标准的 DOM 事件流中,文本结点和 window 都会加入到事件流中。

但是目前 IE 浏览器是不支持事件捕获的。并且即使在支持 DOM 事件捕获的 Firefox 浏览器中,也无法为文本结点添加事件处理函数。

下面的示例可以用于演示事件的捕获阶段和冒泡阶段,注意捕获阶段只能应用于 Firefox 浏览器。

【代码路径: jQueryStorm.Web/chapter7/Demo-DomEvent.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" id="html1">
<head>
  <title>jQuery - DOM 事件模型</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
</head>
<body id="body1">
  <!-- 页面内容部分 -->
  <div id="div1" style="border: solid 1px;"><span id="span1">图层 1</span></div>
  <br />
  <div id="divMsg"></div>
  <!-- 尾部脚本块 -->
  <script type="text/javascript">
```

```
function console(text) {
    $("#divMsg").append("<div>" + text + "</div>");
}

$(function() {
    $("*").each(function() {
        this.addEventListener("click", function(e) {
            console("捕获阶段, " + this.tagName + "#" + this.id + " 事件类型:" + e.type +
" 事件源: " + e.target.id)
        }, true);
        this.addEventListener("click", function(e) {
            console("冒泡阶段, " + this.tagName + "#" + this.id + " 事件类型:" + e.type +
" 事件源: " + e.target.id)
        }, false);
    });

    window.id = "window1";
    window.addEventListener("click", function(e) {
        console("捕获阶段, window" + "#" + this.id + " 事件类型:" + e.type +
" 事件源: " + e.target.id)
    }, true);
    window.addEventListener("click", function(e) {
        console("冒泡阶段, window" + "#" + this.id + " 事件类型:" + e.type +
" 事件源: " + e.target.id)
    }, false);

    var textNode = document.getElementById("span1").firstChild;
    textNode.addEventListener("click", function(e) {
        console("捕获阶段, " + this.tagName + "#" + this.id + " 事件类型:" + e.type +
" 事件源: " + e.target.id)
    }, true);
    textNode.addEventListener("click", function(e) {
        console("冒泡阶段, " + this.tagName + "#" + this.id + " 事件类型:" + e.type +
" 事件源: " + e.target.id)
    }, false);
    });
</script>
</body>
</html>
```

页面运行后, 单击“图层 1”, 会出现如下效果, 如图 7-2 所示。

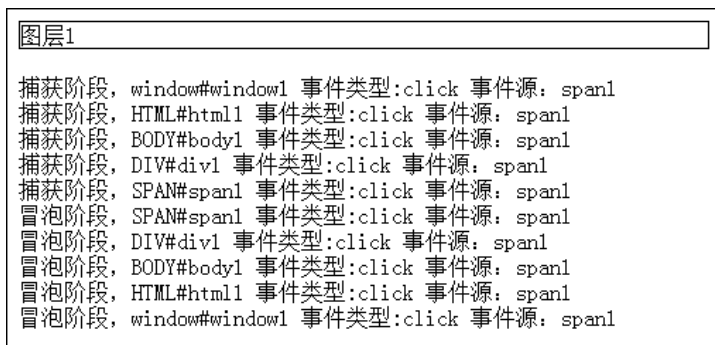


图 7-2 运行效果

通过上面的示例, 可以发现:

- ❑ 使用 jQuery 选择器 “\*” 并不能选中 window 对象。



- ❑ window 对象没有 tagName 属性。
- ❑ DOM 事件流中 window 是根结点，即捕获事件阶段的起始节点、冒泡事件阶段的终结结点。
- ❑ 对于文本结点，也是拥有“addEventListener”方法的，但是即使为文本结点添加了事件，目前在 FireFox 浏览器中也不会触发。

### 7.1.2 事件处理函数

在 DOM 事件模型中，可以为一个事件绑定多个事件处理函数。

在以前的程序中，有时会在 HTML 元素上通过添加属性的方式添加事件处理函数。

```
<div id="testDiv1" onclick="showMsg();">单击事件 1</div>
```

上面的例子为“testDiv1”元素添加了单击时的事件处理函数“showMsg()”。

为了将 HTML 中的行为和结构相分离，也可以使用下面的方式添加事件处理函数。

```
document.getElementById("testDiv2").onclick = showMsg;
```

请注意这两种方式的差异，虽然实现的效果相同，但是这两个函数并不是等价的。

```
<div id="testDiv1" onclick="showMsg();">单击事件 1</div>
```

上面代码的实现方法，相当于创建了一个匿名函数，并且在函数体内调用 showMsg() 方法，也就是和下面的语句是等价的：

```
document.getElementById("testDiv1").onclick = function(event)
{
    showMsg();
};
```

所以要时刻铭记，使用 HTML 属性添加事件，实际上是创建了一个匿名函数。

上面两种方式添加事件处理函数都存在着问题：

- ❑ 只能为一个事件绑定一个事件处理函数，使用“=”赋值会把前面为此事件绑定的所有事件处理函数冲掉。
- ❑ 在事件处理函数（无论是匿名函数还是绑定的函数）中，事件对象在不同浏览器中存在差异。
- ❑ 如果事件处理函数在 DOM 加载完毕之前就已经绑定并触发，可能发生脚本错误。

当事件发生时，会产生“事件对象”，事件对象保存了事件发生时的各种信息。下面会详细讲解事件对象。

为了对一个事件绑定多个事件处理函数，需要使用 DOM 标准提供的 addEventListener() 方法或者 IE 下的 attachEvent() 方法，讨人厌的多浏览器差异又出现了。下面的方法用于处理这种差异：

```
//统一地为对象添加多播事件委托的方法
/*
```

参数说明：

oTarget : 要添加事件的对象.比如"document".

sEventType : 事件类型.比如单击事件"click".

fnHandler : 发生事件时调用的方法. 比如一个静态函数"hideCalendar"



使用举例:

```
//单击页面的任何元素,只要没有取消冒泡,都可以关闭日历控件
ScriptHelper.addEventListener( document, "click", hideCalendar );
```

```
*/
scriptHelper.prototype.addEventListener = function(oTarget, sEventType, fnHandler)
{
    if( oTarget.addEventListener )//for dom
    {
        oTarget.addEventListener( sEventType, fnHandler, false )
    }
    else if( oTarget.attachEvent )//for ie
    {
        oTarget.attachEvent( "on" + sEventType, fnHandler);
    }
}
```

这是传统的处理事件委托的方法，在后面的内容中将会介绍如何使用 jQuery 让一切变得更加简单。

### 7.1.3 事件对象

事件对象是一个事件发生时，保存事件的各种相关信息的对象，比如鼠标单击事件中鼠标的坐标、事件单击的对象、事件是否冒泡、事件的类型等。

然而在 IE 和 DOM 标准的浏览器中，获取事件对象的方法又不相同。在 IE 中，事件对象是 window 对象的一个属性，需要使用下面的方式获取：

```
obj.onclick=function()
{
    var oEvent = window.event;
}
```

在 DOM 标准的浏览器比如 FireFox 中，事件对象作为事件处理函数的唯一参数被传递：

```
obj.onclick=function(e)
{
}
```

或者

```
obj.onclick=function()
{
    var e = arguments[0];
}
```

除了获取事件对象的方式不同，事件对象的属性定义也不相同。

比如在 IE 的事件对象中获取事件源的属性是 e.srcElement，而在 DOM 标准中是 e.target。如表 7-1 所示的是 DOM Level2 标准的事件对象属性。

表 7-1 DOM Level2 事件对象属性

属 性	描 述
bubbles	返回布尔值，指示事件是否是冒泡事件类型
cancelable	返回布尔值，指示事件是否可拥有可取消的默认动作
currentTarget	返回其事件监听器触发该事件的元素
eventPhase	返回事件传播的当前阶段
target	返回触发该事件的元素（事件的目标结点）



(续表)

属 性	描 述
timeStamp	返回事件生成的日期和时间
type	返回当前 Event 对象表示的事件名称

如表 7-2 所示是 DOM Level2 定义的事件对象的方法,具体列出了 2 级 DOM 事件标准定义的方法。IE 的事件模型不支持这些方法。

表 7-2 DOM Level2 事件对象方法

方 法	描 述
initEvent()	初始化新创建的 Event 对象的属性
preventDefault()	通知浏览器不要执行与事件关联的默认动作
stopPropagation()	终止事件在传播过程的捕获、目标处理或冒泡阶段进一步传播。调用该方法后,该结点上处理该事件的处理程序将被调用,事件不再被分派到其他结点

IE 的非标准的事件对象为开发人员造成了不少麻烦。现在使用 jQuery 的事件处理,可以直接轻松地实现跨浏览器的 W3C 标准的事件对象。

## 7.2 jQuery 事件模型

jQuery 让处理事件变得更加简单。通过前面章节的介绍,读者对于标准的 DOM 事件已经有了比较全面的认识。首先介绍 jQuery 中的事件流。

### 7.2.1 jQuery 中的事件流

因为 IE 并不支持事件捕获,所以 jQuery 为了统一事件模型,删除了事件捕获。即只有冒泡过程,如图 7-3 所示。

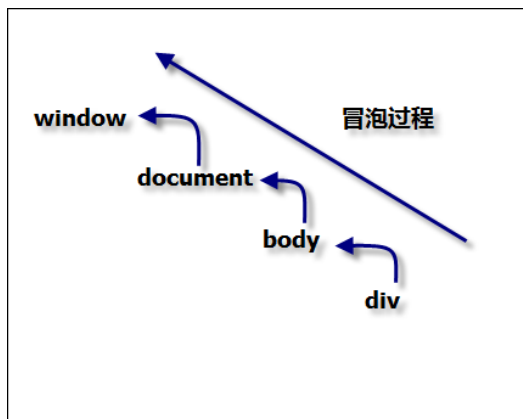


图 7-3 jQuery 事件模型

下面是 jQuery 事件流的例子。

【代码路径: jQuery.Web/chapter7/Demo-JQueryEvent.htm】

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"



```

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" id="html1">
<head>
  <title>jQuery - jQuery 事件模型</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
</head>
<body id="body1">
  <!-- 页面内容部分 -->
  <div id="div1" style="border: solid 1px;"><span id="span1">图层 1</span></div>
  <br />
  <div id="divMsg"></div>
  <!-- 尾部脚本块 -->
  <script type="text/javascript">
    function console(text) {
      $("#divMsg").append("<div>" + text + "</div>");
    }
    $(function() {
      $("*").click(function(e) {
        console(this.tagName + "#" + this.id + " event target: " + e.target.id);
      });
      $(window).click(function(e) {
        console(this.tagName + "#" + this.id + " event target: " + e.target.id);
      });
    });
  </script>
</body>
</html>

```

在上面的实例中，使用 jQuery 的选择器 “\*” 选中了除 window 对象以外的所有对象，为其添加了单击事件。对于 window 也可以使用 jQuery 的事件绑定函数，同样添加了 click 事件。

当单击页面的“图层 1”文字后，效果如下：

图层1

```

SPAN#span1 event target: span1
DIV#div1 event target: span1
BODY#body1 event target: span1
HTML#html1 event target: span1
undefined#undefined event target: span1

```

从结果中可以看出，使用 jQuery 的事件绑定函数添加事件，只作用在冒泡阶段。这种事件模型使得事件流在所有浏览器得到统一。

使用 jQuery 的事件机制，具有如下好处：

- ❑ 统一事件模型，在所有浏览器中使用统一的冒泡事件模型。
- ❑ 统一事件名称，在 IE 下使用 attachEvent 添加多播事件时要使用“onclick”，而标准 DOM 是“click”。在 jQuery 事件模型中统一使用“click”添加单击事件。
- ❑ 统一事件绑定函数，能够为单个事件添加多个事件处理函数（多播事件委托）。比如下面的例子为 div 的单击事件添加了两个事件处理函数：

```

$("#testDiv4").bind("click", function(event) { alert("one"); });
$("#testDiv4").bind("click", function(event) { alert("two"); });

```

- ❑ 统一事件对象，使用 jQuery 可以在 IE 中也使用 DOM 标准的事件对象。





在 jQuery 函数分类中，“Event”分类用于放置所有和事件相关的函数。Event 分类下又有很多的子分类，如表 7-3 所示。

表 7-3 Event 子分类列表

分类名称	分类说明
Browser Events	和浏览器相关的事件函数，比如 error 错误、窗口大小调整、滚动条滚动等
Document Loading	页面加载相关的事件函数，比如 load()、ready()、unload()等
Event Handler Attachment	事件绑定函数，比如最基础的 bind()和 unbind()，监听 DOM 的 live()和 die()等
Event Object	事件对象说明，包括事件对象的所有属性和方法
Form Events	表单相关的快捷函数
Keyboard Events	键盘相关的快捷函数
Mouse Events	鼠标相关的快捷函数

下面将分别介绍 jQuery 的事件绑定函数和事件对象。

## 7.2.2 jQuery 事件绑定函数

jQuery 的事件绑定函数用于为对象添加事件和事件处理函数。使用 jQuery 的事件绑定函数是多浏览器兼容的，因此以后就可以忽略浏览器对于事件绑定的差异，让页面开发回归简单。

使用 jQuery 绑定事件处理函数主要有两种方法。一种是使用 bind()和 unbind()，需要传递事件的名称。另一种是使用快捷事件，比如.click()，不需要传递事件名称。

下面是这两种方法的举例：

```
$("#div1").bind("click", function(e) { console("单击了图层 1"); });
$("#div2").click(function(e) { console("单击了图层 2"); });
```

上面的例子都为 div 对象添加了单击事件。事件处理函数的第一个参数是 DOM 标准的事件对象。

接下来详细讲解这两种事件绑定方式。

### 1. 使用.bind()和.unbind()函数添加和删除事件绑定

.bind()函数用于绑定事件处理函数，.unbind()函数用于删除事件处理函数。这两个函数位于 Event-> Event Handler Attachment 分类中。

下面是这两个函数的所有函数重载，如表 7-4 所示。

表 7-4 bind()和 unbind()函数重载列表

函数签名	返回值	版 本	函数说明	参数说明
.bind( eventType, [ eventData ], handler (eventObject) )	jQuery	1.0	为每个匹配元素的特定事件绑定事件处理函数	eventType: 事件类型名称 eventData: 可选参数。事件数据。 handler(eventObject): 事件处理函数，eventObject 为事件对象
.bind( events )	jQuery	1.4	一次绑定多个事件。传入一个名值对对象，名为事件名称，值为事件处理函数	attributeName: 属性名称 value: 属性值





(续表)

函数签名	返回值	版 本	函数说明	参数说明
<code>.unbind( eventType, handler(eventObject) )</code>	jQuery	1.0	<p><code>bind()</code>的反向操作, 从每一个匹配的元素中删除绑定的事件。</p> <p>如果没有参数, 则删除所有绑定的事件。</p> <p>可以将 <code>bind()</code>注册的自定义事件取消绑定。</p> <p>如果提供了事件类型作为参数, 则只删除该类型的绑定事件</p> <p>如果把在绑定时传递的处理函数作为第二个参数, 则只有这个特定的事件处理函数会被删除</p>	<p><code>eventType</code>: 事件类型名称</p> <p><code>handler(eventObject)</code>: 事件处理函数, <code>eventObject</code> 为事件对象</p>
<code>.unbind( event )</code>	jQuery	1.0	<p><code>bind()</code>的反向操作, 从每一个匹配的元素中删除绑定的事件</p>	<p><code>event</code>: 事件对象</p>

在事件相关的方法中, 常常用到 `eventType` 参数, `eventType` 是事件类型名称, `eventType` 可以是 JavaScript 的事件名称, 比如 `click`、`mouseover` 等, 如果是 JavaScript 能够识别的事件名称, 则会绑定到相应的浏览器事件上。比如 “`click`” 表示鼠标单击事件。`eventType` 还可以绑定用户自定义事件, 用户自定义事件不会由浏览器触发, 而需要使用 `.trigger()` 或 `.triggerHandler()` 方法触发。

下面是最典型的 `.bind()` 函数的应用:

```
$("#div1").bind("click", function(e) { console("单击了图层 1"); });
```

(1) 使用命名空间。

`eventType` 参数使用 “.” 字符为事件添加命名空间, 比如:

```
$("#div1").bind("click.myName", function(e) { console("触发了命名空间为 myName 的单击事件"); });
$("#div1").bind("click.yourName", function(e) { console("触发了命名空间为 yourName 的单击事件"); });
```

其中 `click` 是指事件类型, `myName` 和 `yourName` 是命名空间。

使用了命名空间后, 仍然可以使用平常方式触发事件或者解除事件绑定:

```
$("#div1").unbind("click");
```

使用上面的方法将删除为元素绑定的所有单击事件处理函数。如果想只删除其中的某一个事件处理函数要怎么做呢? 如果保留了事件处理函数的引用, 可以使用 `.unbind( eventType, handler(eventObject) )` 这个重载为单击事件删除某一个事件处理函数。但是实例中因为使用的是匿名函数, 没有保存事件处理函数的引用, 所以无法做到这一点。此时就可以使用命名空间达到相同的效果:

```
$("#div1").unbind("click.yourName");
```

上面的代码删除了 `yourName` 命名空间下的事件处理函数, 而 `myName` 命名空间下的事件处理函数依然起作用。

除了可以删除单独的事件处理函数, 还可以在多个事件上使用相同的命名空间, 然后一次删除多个事件的事件处理函数:



```
$("#div1").bind("click.myName", function(e) { console("图层 1 触发了命名空间为 myName 的单击事件"); });
$("#div1").bind("dblclick.myName", function(e) { console("图层 1 触发了命名空间为 myName 的双击事件"); });
$("#div1").unbind(".myName");
```

上面的代码同时删除了 click 和 dblclick 事件的 myName 命名空间下的事件处理函数。

## (2) 多事件绑定。

在 1.4 版本中提供了新的签名方法.bind( events )用于一次绑定多个事件:

```
$("#div1").bind({
  click: function() {
    //添加单击事件
  },
  dblclick: function() {
    //添加双击事件
  }
});
```

上面的代码同时为 ID 为 div1 的图层绑定了单击和双击事件。

如果两个事件的事件处理函数相同，还可以改成:

```
$("#div1").bind("click dblclick", function(e) { console("事件类型: "+e.type); });
```

通过事件对象 e 可以获取到触发的事件类型及事件对象。有关事件对象的使用技巧将在下面讲解。

## 2. 使用快捷事件绑定函数

jQuery 提供了常用事件的快捷函数，比如 click()，来快速地为对象绑定事件。

以.click()函数为例。

使用.click()函数绑定事件:

```
.click( handler(eventObject) );
```

相当于

```
.bind('click', handler(eventObject));
```

使用.click()函数触发事件:

```
.click();
```

相当于

```
.trigger('click');
```

如表 7-5 到 7-9 所示的是在 Event 一级分类下，各二级分类的快捷函数列表。

表 7-5 Event->Browser Events 分类函数列表

函数名称	函数说明	函数举例
.error()	在大多数浏览器中，当页面的 JavaScript 发生错误时，window 对象会触发 error 事件；当图像的 src 属性无效时，比如文件不存在或者图像数据错误时，也会触发图像对象的 error 事件。 error 事件不会冒泡。 window 的 error 事件在 IE 8 和 Chrome 中失效	在服务器端记录 JavaScript 错误日志: \$(window).error(function(msg, url, line){ jQuery.post("js_error_log.aspx", { msg: msg, url: url, line: line });}); 隐藏 JavaScript 错误: \$(window).error(function(){ return true;



(续表)

函数名称	函数说明	函数举例
		}); 如果图像加载不到, 设置成默认图像: \$("img").error(function(e) { this.src = "../pic/default.jpg"; });
.resize()	浏览器重置窗口大小事件。 需要注意, 如果是拖动改变浏览器的窗口大小, 则会频繁地触发 resize 事件	在窗口改变大小时, 在页面上显示窗口大小: \$(window).resize(function() { \$('body').prepend('<div>' + \$(window).width() + '</div>'); });
.scroll()	当滚动条发生变化时触发。 滚动事件可以发生在 window 对象上, 也可以发生在 iframe 或者某一元素上	为浏览器滚动绑定事件处理函数: \$(window).scroll( function() { /* ...do something... */ } );

表 7-6 Event->Document Loading 分类函数列表

函数名称	函数说明	函数举例
.load()	如果绑定给 window 对象, 则会在所有内容加载后触发, 包括窗口、框架、对象和图像。如果绑定在元素上, 则当元素的内容加载完毕后触发。 注意: 只有当在这个元素完全加载完之前绑定 load 的处理函数, 才会在它加载完后触发。如果之后再绑定就永远不会触发了。所以不要在\$(document).ready()里绑定 load 事件, 因为 jQuery 会在所有 DOM 加载完成后绑定 load 事件	页面加载完毕后, 为所有宽度大于 100 像素的图片添加 bigImg 样式: \$('img').load(function(){ if(\$(this).width() > 100) { \$(this).addClass('bigImg'); } });
.ready()	只有 document 有 ready 事件, 此事件作用在所有 DOM 元素比如图片的 load 事件之前, 在 DOM 加载完毕后执行。 有关 ready 的详细使用, 在“jQuery 核心”一节有详细讲解	使用三种方式添加页面处理程序: \$(document).ready(handler); \$.ready(handler) (this is not recommended); \$(handler);
.unload()	只有 window 对象有 unload 事件, 在关闭选项卡窗口、单击超链接、在地址栏输入 url 等都会触发 unload 事件	关闭窗口或离开页面时提示用户: \$(window).unload( function () { alert("欢迎再来"); } );

表 7-7 Event->Form Events 分类函数列表

函数名称	函数说明	函数举例
.blur()	失去焦点事件	\$("p").blur();
.change()	当表单元素内容变化时触发的事件。以下元素可以绑定 change 事件: <input type="text">、<textarea>、<select> 只有 select 元素改变选项时会立刻触发 change 事件, 其他元素在失去焦点时触发	\$('#mySelect').change(function() { alert(\$(this).val()); });
.focus()	获取焦点事件	\$('#target').focus(function() { alert("获取了焦点"); });



(续表)

函数名称	函数说明	函数举例
.select()	选中文本框或文本区域的文字时触发的事件	<pre> \$("input").select( function () {     \$("div").text("Something was selected").show().     fadeOut(1000); }); </pre>
.submit()	表单提交事件	<pre> \$('#other').click(function() {     \$('#target').submit(); }); </pre>

表 7-8 Event->Keyboard Events 分类函数列表

函数名称	函数说明	函数举例
.focusin()	自己获取到焦点或者其子元素获取到焦点时触发	<pre> \$("input"). focusin (function() {     //do something }); </pre>
.focusout()	自己失去焦点或者其子元素失去焦点时触发	<pre> \$("input"). focusout (function() {     //do something }); </pre>
.keydown()	键盘按下事件	<pre> \$("input"). keydown (function() {     //do something }); </pre>
.keypress()	和 keydown 类似，但是当按住一个键不放时，只触发一次 keydown 事件，却会触发多次 keypress 事件	<pre> \$("input").keypress(function() {     //do something }); </pre>
.keyup()	键盘释放事件	<pre> \$("input"). keyup (function() {     //do something }); </pre>

表 7-9 Event->Mouse Events 分类函数列表

函数名称	函数说明	函数举例
.click()	单击事件	<p>将页面内所有段落单击后隐藏:</p> <pre> \$("p").click( function () { \$(this).hide(); } ); </pre>
.dblclick()	双击事件	<p>给页面每个段落的双击事件绑上“Hello World!”警告框:</p> <pre> \$("p").dblclick( function () { alert("Hello World!"); } ); </pre>
.hover()	改进的 mouseover 事件，hover 函数是对 mouseenter 和 mouseleave 的封装调用	见本章后续详解
.mousedown()	鼠标按键按下事件	<p>当鼠标按下时显示提示:</p> <pre> \$("p").mousedown(function(){ \$(this).append("mouse down"); }); </pre>



(续表)

函数名称	函数说明	函数举例
.mouseup()	鼠标按键释放事件	当鼠标按键释放时显示提示: <code>\$( "p" ).mouseup(function(){  \$(this).append("mouse up");  });</code>
.mouseenter()	与 mouseover 的区别是鼠标滑动到元素的子元素时不会再次触发绑定元素的事件	当鼠标移动到元素上时显示提示: <code>\$( "div" ).mouseenter(function(){  \$(this).append("mouse enter");  });</code>
.mouseleave()	与 mouseout 的区别是在绑定元素的子元素上离开时不会触发此事件	当鼠标从元素上离开时显示提示: <code>\$( "div" ).mouseleave(function(){  \$(this).append("mouse leave");  });</code>
.mousemove()	鼠标在绑定元素上移动时触发的事件, 每移动一个像素就触发一次	当鼠标移动时显示鼠标的坐标: <code>\$( "div" ).mousemove(function(e){  var pageCoords = "( " + e.pageX + ", " + e.pageY + " )";  var clientCoords = "( " + e.clientX + ", " + e.clientY + " )";  \$( "span:first" ).text("( e.pageX, e.pageY ) - " + pageCoords);  \$( "span:last" ).text("( e.clientX, e.clientY ) - " + clientCoords);  });</code>
.mouseout()	鼠标离开元素触发的事件	当鼠标从绑定元素或其子元素上离开时显示提示: <code>\$( "div" ).mouseout(function(){  alert("mouse out");  });</code>
.mouseover()	鼠标移动到元素上时触发的事件	当鼠标移动到绑定元素或其子元素上时显示提示: <code>\$( "div" ).mouseover(function(){  alert("mouse over");  });</code>

在上面的列表中, 有一些事件并不是标准事件, 而是 jQuery 函数库提供的特有的改进事件, 将在后面做详细讲解。

### 7.2.3 事件处理函数中的 this 指针

在第 1 章介绍的例子中, 使用为一个页面建立一个脚本对象的方式组织页面脚本。回忆一下 thisPage 类:

```
var thisPage = {
  initialize: function () { //加载时执行
    this.initializeDom();
    this.initializeEvent();

    //加载时隐藏 divMsg 图层
    this.$divMsg.hide(300);
  },
  initializeDom: function () { //初始化 DOM
    this.$divMsg = $("#divMsg");
```



```

        this.$btnShow = $("#btnShow");
    },
    initializeEvent: function () { //事件绑定
        this.$btnShow.bind("click", function (event) {
            $("#divMsg").toggle(300);
        });
    }
}

```

上面是第 1 章“Hello jQuery!”的例子。在 `thisPage` 类中，`this` 指针是指向 `thisPage` 对象的。所以在 `initializeDom` 事件中为 `this` 的 `$divMsg` 和 `$btnShow` 属性赋值，然后在 `initializeEvent` 中使用 `this.$btnShow` 属性。当时就提到在事件处理函数中不能使用 `this.$divMsg`，是因为这里仅仅是事件处理函数，是在用户触发了事件后被调用的，在第 2 章中讲解了 `this` 指针，`this` 指针总是指向函数的调用者。所以事件处理函数中的 `this` 指针是触发事件的 DOM 对象本身。

要解决此问题，将事件处理函数中的 `this` 指针修改成指向 `thisPage` 对象，可以使用 `jQuery.proxy` 函数：

```
this.$btnShow.bind("click", jQuery.proxy(function (event) { this.$divMsg.toggle(300); }, this));
```

`jQuery.proxy` 返回的是一个 `function`，并且此 `function` 中的 `this` 已经被修改为指向 `thisPage` 对象。`jQuery.proxy()` 函数接受两个参数，第一个是待处理的 `function` 对象，第二个是 `this` 指针指向的对象。上面的示例语句运行时，传递给 `jQuery.proxy` 的第二个参数“`this`”是指向 `thisPage` 对象的。有关 `jQuery.proxy` 的详细说明和使用方法，请参见第 10 章的内容。

`jQuery.proxy()` 的一个问题就是语法不够优美，因为它是工具函数，只能通过 `jQuery` 对象来使用。所以通过扩展 `Function` 的原型，我们有自己的实现：

```

jQuery.extend(Function.prototype, {
    bind: function(context)
    {
        var method = this;
        var args = jQuery.makeArray(arguments);
        var obj = args.shift();
        return function(event)
        {
            return method.apply(obj, [event].concat(args));
        }
    }
});

```

现在可以使用“链式操作”来修改函数的 `this` 了：

```
this.$btnShow.bind("click", function (event) { this.$divMsg.toggle(300); }.bind(this) );
```

这个函数将会在第 13 章中详细介绍。

#### 7.2.4 jQuery 事件对象

上面已经介绍过 DOM 标准的事件对象，因为 IE 的事件对象和 DOM 标准是不同的，所以 `jQuery` 对事件对象做了处理，对事件对象进行了格式化，使得在包含 IE 在内的所有浏览器中都能够使用 DOM 标准的事件对象。

事件对象是作为第一个参数传递给事件处理函数的，在本书及 `jQuery` API 中，常常看到绑定的事件处理函数的签名是这样的：



```
.click( handler(eventObject) )
.bind( eventType, [ eventData ], handler(eventObject) )
```

上面的 **handler** 是事件处理函数，其中的参数 **eventObject** 是事件对象，作为第一个参数传递给事件处理函数。当然在实际使用中，可以为事件对象起一个更短的变量名：

```
$("#p").click(function(e) {
    //e 是事件对象
});
```

但是如果使用 **.trigger()** 函数在脚本段触发事件而不是由用户触发，那么事件对象会有很多的属性和方法是无法使用的。jQuery 提供了编程创建事件对象的构造函数：

```
var e = new jQuery.Event("click");
```

或者可以省略 **new**：

```
var e = new jQuery.Event("click");
```

jQuery 的事件对象的属性及方法如表 7-10 和表 7-11 所示。

表 7-10 jQuery 事件对象属性

属 性	说 明
currentTarget	在事件冒泡时触发当前事件的 DOM 对象。总是和事件处理函数中的“this”相当： <pre>\$("#p").click(function(event) {     alert( event.currentTarget === this ); // true });</pre>
relatedTarget	非标准 DOM 属性。事件的另外一个关联对象，对于 mousein、mouseout 一类的事件，需要涉及两个对象（比如从 a 移动到 b），其中事件绑定的对象是 currentTarget，另外一个关联对象是 relatedTarget。比如为一个 div 绑定了 mousein 事件，则事件发生时 currentTarget 是这个 div，relatedTarget 是 body（假设 div 在 body 中）。对于 click 这种只和一个对象有关的事件，relatedTarget 是 undefined，引用的话会发生错误
target	返回触发此事件的元素（事件的目标结点）。因为事件会冒泡，所以当触发父元素的事件时，target 是不变的，指向原始的触发事件的对象，而 currentTarget 会随着冒泡而改变
timeStamp	返回事件生成的日期和时间。自 1970 年 1 月 1 日开始的毫秒数
type	返回当前 Event 对象表示的事件名称
data	非标准 DOM 属性。当使用 bind 绑定事件处理函数时，可以传递一个可选的 data 参数，此参数通过事件对象的数据属性引用： <pre>\$("#a").each(function(i) {     \$(this).bind('click', {index:i}, function(e){         alert('my index is ' + e.data.index);     }); });</pre>
pageX	鼠标位置相对于文档左上角的 x 坐标值
pageY	鼠标位置相对于文档左上角的 y 坐标值
result	非标准 DOM 属性，最近一次绑定了此事件的事件处理函数的返回值。如果没有最近的事件处理函数返回值则是 undefined。 不如为 click 事件绑定两个事件处理函数，第一个返回内容： <pre>\$("#div2").click(function(event) { return "hello" }); \$("#div2").click(function(event) { alert(event.result); });</pre> 则在第二个事件处理函数中 event.result 的值为 hello





(续表)

属 性	说 明
which	如果是鼠标事件,则表示鼠标按键的按键代码:左键为 1、中键为 2、右键为 3, 如果是键盘事件则为键盘码,比如 a 的键盘码为 65

表 7-11 jQuery 事件对象方法

方 法	说 明
initEvent()	初始化新创建的 Event 对象的属性
preventDefault()	通知浏览器不要执行与事件关联的默认动作
stopPropagation()	终止事件在传播过程的捕获、目标处理或冒泡阶段进一步传播。调用该方法后,该结点上处理该事件的处理程序将被调用,事件不再被分派到其他结点
isDefaultPrevented()	返回一个布尔值,表示是否调用了 preventDefault()方法
isImmediatePropagationStopped()	返回一个布尔值,表示是否调用了 stopImmediatePropagation()方法
isPropagationStopped()	返回一个布尔值,表示是否调用了 stopPropagation()方法

可以发现, jQuery 的事件对象与标准的 DOM 事件对象还是有区别的,比如没有格式化 bubbles、cancelable 属性,而是使用另外的方法 .isPropagationStopped() 和 .isImmediatePropagationStopped() 替代。注意这里提到的没有格式化,是指没有将此属性做特殊处理,在实现 DOM 标准的浏览器中,比如 FireFox、bubbles 等属性都是可以使用的,但是在 IE 中就无法使用。

### 1. 获取事件相关的元素

在事件处理中,常常要操作和事件相关的元素。事件对象有三个和对象相关的属性,分别是 target、currentTarget 和 relatedTarget。

target 是触发事件的元素,即使因为事件冒泡触发了父元素的事件,在所有冒泡的事件处理函数中 target 的值都是相同的,而 currentTarget 的值是不同的。下面例子可以很好地说明这三者的不同:

```
【代码路径: jQueryStorm.Web/chapter7/Demo-EventObjTarget.htm】
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" id="html1">
<head>
  <title>jQuery - jQuery 事件对象</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
  <style>
    div {border: solid 1px; margin:5px;}
  </style>
</head>
<body id="body1">
  <!-- 页面内容部分 -->
  <div id="outer">
    outer
    <div id="inner">
      inner
    </div>
  </div>
  <br />
  <div id="divMsg"></div>
  <!-- 尾部脚本块 -->
```





```
<script type="text/javascript">
    function console(text) {
        $("#divMsg").append("<div>" + text + "</div>");
    }
    $(function() {
        $("div").not("#divMsg").click(function(e) {
            //输出 target、currentTarget 和 relatedTarget
            console("target: " + e.target.id + ", currentTarget: " + e.currentTarget.id + ", relatedTarget: " +
e.relatedTarget);
        });
    });
</script>
</body>
</html>
```

当单击 inner 区域时，结果如图 7-4 所示。

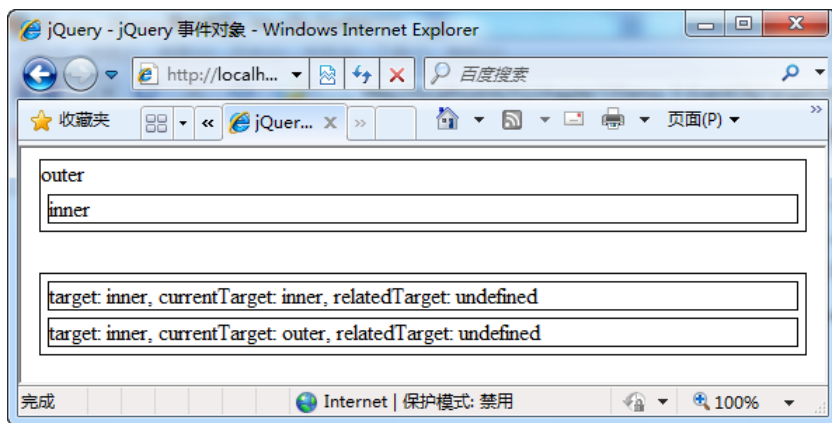


图 7-4 事件对象中的元素

事件首先在 inner 中被触发，然后冒泡到 outer，所以会有两条记录，分别触发了 inner 和 outer 的事件处理函数，在这两个事件处理函数中，事件对象的 target 始终指向事件触发的元素 inner，事件对象的 currentTarget 则指向当前事件处理函数绑定的对象，而 relatedTarget 和 undefined 是因为没有额外的关联元素。什么时候才会有 relatedTarget 属性呢？比如 mouseout 事件，鼠标移动出来后的对象就是 relatedTarget 对象。对于 mousein 事件，鼠标移动进去的元素就是 relatedTarget 元素。也就是说 mouseout 和 mousein 这种事件会关联两个对象，事件触发对象是事件源 target，而目标对象就是 relatedTarget。

## 2. 取消事件冒泡

通过事件对象的 stopPropagation()方法，可以取消事件冒泡，比如修改上面的实例：

```
$(function() {
    $("div").not("#divMsg").click(function(e) {
        console("target: " + e.target.id + ", currentTarget: " + e.currentTarget.id + ", relatedTarget: " +
e.relatedTarget);
        e.stopPropagation();
    });
});
```

此时单击 inner 是，不会再触发 outer 的 click 事件，结果如图 7-5 所示。



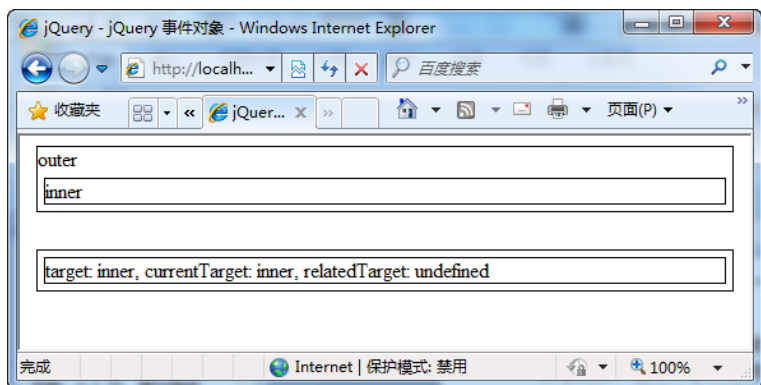


图 7-5 取消事件冒泡

通过 `isPropagationStopped()` 方法，还可以获取到当前的事件是否已经取消了冒泡行为：

```
$( "p" ).click( function( event ) {
    alert( event.isPropagationStopped() );
    event.stopPropagation();
    alert( event.isPropagationStopped() );
});
```

除了 DOMLevel2 标准的 `stopPropagation()` 方法，jQuery 还提供了 DOM Level3 标准的 `stopImmediatePropagation()` 方法。调用 `stopImmediatePropagation()` 方法后，不仅仅会阻止事件冒泡，而且会立刻阻止此事件的其他事件处理函数的运行。

比如使用 `stopPropagation()` 函数：

```
$( "div" ).not( "#divMsg" ).click( function( e ) {
    console( "1" );
    e.stopPropagation();
});
$( "div" ).not( "#divMsg" ).click( function( e ) {
    console( "2" );
});
```

上面的例子为 `div` 绑定了两个 `click` 事件的事件处理函数，分别输出 1 和 2。当单击 `div` 时，虽然已经取消了事件冒泡，但是当前元素的两个事件处理函数都会执行，输出结果为：

```
1
2
```

而如果改成使用 `stopImmediatePropagation()` 方法，则只会执行一个事件处理函数，即指输出 “1”：

```
$( "div" ).not( "#divMsg" ).click( function( e ) {
    console( "1" );
    e.stopImmediatePropagation(); //会阻止输出 2 的事件处理函数运行
});
$( "div" ).not( "#divMsg" ).click( function( e ) {
    console( "2" );
});
```

这一切都是 jQuery 内部实现的，目前还少有浏览器原生支持 DOM Level3 的此方法。

可以通过 `isImmediatePropagationStopped()` 函数，获取是否已经执行了 `stopImmediatePropagation()` 方法：

```

$("div").not("#divMsg").click(function(e) {
    console("1");
    console(e.isImmediatePropagationStopped()); //输出为:false
    e.stopImmediatePropagation();
    console(e.isImmediatePropagationStopped()); //输出为: true
});

```

在 jQuery 内部阻止其他事件处理函数执行时，也是使用 "isImmediatePropagationStopped()" 函数判断的。

### 3. 取消浏览器默认行为

当单击一个带有 href 属性的 a 元素时，会打开超链接，但是并没有手动地为 a 元素绑定单击事件，这种元素默认的行为称为浏览器默认行为。

同样具有浏览器默认行为的还有表单中的提交按钮。如果表单中有提交按钮：

```

<form method="post" action="test.html">
  <input type="submit" value="提交按钮" />
</form>

```

则单击此按钮时会提交表单。

DOM 标准的事件对象提供了 preventDefault() 方法用来取消元素的默认行为，jQuery 中也提供了此方法。下面的例子为所有的 a 元素和提交按钮取消了默认行为：

```

$("a").click(function(e) {
    e.preventDefault(); //返回 false;
});

$("input:submit").click(function(e) {
    e.preventDefault(); //返回 false;
});

```

在 jQuery 的事件处理函数中，preventDefault() 方法是跨浏览器通用的，这是因为 jQuery 内部对这一方法做了处理。IE 浏览器不支持 “preventDefault()” 方法，但是可以通过返回 “false” 的方法取消其默认行为。所以在 jQuery 内部的 preventDefault() 方法会先判断浏览器是否原生地支持 preventDefault() 方法，如果不支持则使用返回 “false” 的方式。这两种方式都可以实现阻止元素的默认行为的效果：

```

$("a").click(function(e) {
    e.preventDefault();
});

```

或者

```

$("a").click(function(e) {
    return false;
});

```

下面是 jQuery 的内部实现：

```

preventDefault: function() {
    this.isDefaultPrevented = returnTrue;
    var e = this.originalEvent;
    if ( !e ) {
        return;
    }
    //如果 preventDefault 事件是浏览器原生支持的，则直接调用
    if ( e.preventDefault ) {

```





```
e.preventDefault();
}
e.returnValue = false;    //否则使用返回 false 值的方式（在 IE 中）
}
```

## 7.3 jQuery 特殊事件

在 jQuery 的事件处理机制中，提供了一些特殊的事件函数，实现 JavaScript 原生事件所不具备的功能。本节将集中介绍这些事件。

### 7.3.1 对象监听函数 live 和 die

live()和 die()函数在 jQuery 1.3 版本就已经提供，但是当时存在着很多问题。在 1.4 版本中，live()和 die()函数被优化并且改进，已经达到了可用的状态，如表 7-12 所示。

表 7-12 live 和 die 函数签名列表

函数签名	返回值	版 本
.live( eventType, handler )	jQuery	1.3
.live( eventType, eventData, handler )	jQuery	1.4
.die()	jQuery	1.4.1
.die( eventType, [ handler ] )	jQuery	1.3

live()和 die()函数也被叫做“事件委托”，但是就其实起作用而言更应该称做“对象监听函数”。通过 live()绑定事件处理函数，不仅为当前选中的元素添加了事件处理函数，如果未来页面上添加了新的元素并且匹配第一次调用 live()时元素的选择器表达式，则新添加的元素也会具有 live()绑定的事件，相当于监听了对象变更。

比如为所有样式为“clickclass”的元素添加单击事件，使用 bind()方法：

```
$('.clickclass').bind('click', function() {
});
```

上面的代码页面上所有的样式为 clickclass 元素添加了 click 事件处理函数。

如果在上面的语句执行完毕后，页面添加了新的元素：

```
$('#body').append('<div class="clickclass">Another target</div>');
```

虽然新添加的对象也应用了“clickclass”样式，但是并没有为其绑定 click 事件处理函数。而如果使用 live 则可以实现此功能：

```
$(".clickclass").live('click', function() {
    //添加 live 事件处理函数
});
```

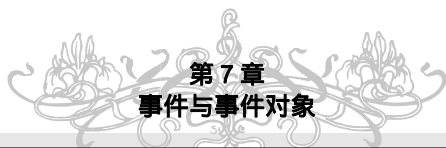
现在，无论是原有元素，还是新添加的元素，只要应用了“clickclass”样式，都会被绑定为 click 事件。

#### 1. eventData 事件数据

在使用三个参数的 live 函数时：

```
.live( eventType, eventData, handler )
```





eventType 是事件名称、eventData 是传递给事件处理函数的额外数据、handler 是事件处理函数，这些都和 bind()方法完全一致。

“eventData”参数可以通过事件对象的数据属性获取：

```
$(".clickclass").live("click", "myData", function(e) { alert(e.data ); });
```

上面的代码中，传入的“myData”字符串可以通过事件对象的“data”属性“e.data”获取到。

## 2. 绑定多事件

jQuery 1.4 版本后，live()函数可以一次添加多个事件：

```
$(".clickclass").live("click mouseleave", function(e) {  
    alert(this.id + " " + e.type);  
});
```

## 3. 事件上下文

jQuery 1.4 版本后，可以设置监听对象变化的范围，比如只监听某一个 div 中的对象变化：

```
$(".clickclass", "#divContainer").live("click", clickHandler1);
```

现在只有在 ID 为“divContainer”的 div 中，添加应用了“clickclass”样式的对象时，才会为对象添加单击事件。

此功能是通过传递 jQuery 选择器的上下文实现的。

## 4. 核心实现

live()和 die()的实现其实并不复杂。当使用 live()为某一个上下文添加了事件，比如 click 事件，实际上并没有为每一个匹配元素添加 click 事件，而是在上下文对象，即默认的 document 上，添加了一个 click 事件的事件处理函数。当单击新添加的元素时，会进行如下处理：

- (1) 调用元素使用 bind()绑定 click 事件处理函数。
- (2) 事件冒泡到使用 live()绑定时的上下文对象上时，调用使用 live()绑定的事件处理函数。
- (3) 在事件处理函数中，判断“\$(event.target).closest(selector,[context])”能否找到匹配的对象。
- (4) 如果找到了对象，则为此对象调用此事件处理函数。

## 5. 删除对象监听函数

使用 live()添加的函数可以使用 die()删除：

```
$(".clickclass").die();
```

无参数的 die()重载将删除所有的对象监听函数。

如果为某一种选择器添加了多个事件的对象监听函数，则可以传递 eventType 删除某一个事件的对象监听函数：

```
$(".clickclass").die("mouseleave");
```





如果在某一个事件上添加了多个事件处理程序，也可以只删除特定的事件处理程序：

```
$(".clickclass").live("click", clickHandler1);
$(".clickclass").live("click", clickHandler2);
$("body").append("<div id='div2' class='clickclass'>元素 2</div>");
$(".clickclass").die("click", clickHandler1);
```

## 6. 注意事项

虽然 live() 的用法和 bind() 十分接近，但两者是不同的，这些不同包括：

❑ 集合转换类的方法在 live() 中是不完全被支持的，所以最佳用法是直接在 jQuery 选择器后调用 live()。

❑ 在 1.3 版本的 jQuery 中，live() 只支持如下事件：

Click、dblclick、keydown、keypress、keyup、mousedown、mousemove、mouseout、mouseover 和 and mouseup。

❑ 如果想阻止 live() 绑定的事件处理函数，需要返回 “false”：

```
var clickHandler1 = function(e) { alert("1"); return false; };
var clickHandler2 = function(e) { alert("2"); };
$(".clickclass").live("click", clickHandler1);
$(".clickclass").live("click", clickHandler2);
```

上面的代码就不会继续输出“2”。注意在这里使用事件对象的 stopImmediatePropagation() 方法不起作用：

```
var clickHandler1 = function(e) { alert("1"); e.stopImmediatePropagation(); };
var clickHandler2 = function(e) { alert("2"); };
$(".clickclass").live("click", clickHandler1);
$(".clickclass").live("click", clickHandler2);
```

如果是 bind 绑定的多个事件处理函数，是可以通过 stopImmediatePropagation() 阻止后续事件处理函数继续调用的，但是 live() 绑定的不可以。

❑ 在 1.4 版本中 live() 支持了自定义事件，1.4.1 版本中开始支持 focus 和 blur 事件，但是会映射到 “focusin” 和 “focusout” 这个 jQuery 特殊的事件上。

## 7.3.2 改进的鼠标事件 mouseenter、mouseleave 和 hover

mouseenter 和 mouseleave 事件分别表示鼠标移动到元素上或移出元素，类似于 Javascript 原生的 mouseover 和 mouseout 事件。mouseenter 和 mouseleave 事件是为了弥补 mouseover 和 mouseout 事件的不足。看下面的例子：

【代码路径：jQueryStorm.Web/chapter7/Demo.mouseEnterLeave.htm】

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" id="html1">
<head>
  <title>jQuery - mouseenter 和 mouseleave</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
  <style>
    div.out
    {
      width:40%;
      height:120px;
      margin:0 15px;
      background-color:#D6EDFC;
```



```

        float:left;
    }
    div.in {
        width:60%;
        height:60%;
        background-color:#FFCC00;
        margin:10px auto;
    }
</style>
</head>
<body id="body1">
    <div id="enterleave" class="out">mouseenter / mouseleave<div class="in"></div></div>
    <div id="overout" class="out">mouseover / mouseout<div class="in"></div></div>
    <br style="clear:both;" />
    <div id="divMsg1">MouseEnter:<span id="enter">0</span>,
MouseLeave:<span id="leave">0</span></div>
    <div id="divMsg2">MouseOver:<span id="over">0</span>, MouseOut:<span id="out">0</span></div>
    <!-- 尾部脚本块 -->
    <script type="text/javascript">
        $(function() {
            $("#enterleave").mouseenter(function() { //添加 mouseenter 事件
                var n = parseInt($("#enter").text()) + 1;
                $("#enter").text(n);
            }).mouseleave(function() { //添加 mouseleave 事件
                var n = parseInt($("#leave").text()) + 1;
                $("#leave").text(n);
            });

            $("#overout").mouseover(function() { //添加 mouseover 事件
                var n = parseInt($("#over").text()) + 1;
                $("#over").text(n);
            }).mouseout(function() { //添加 mouseout 事件
                var n = parseInt($("#out").text()) + 1;
                $("#out").text(n);
            });

            $("#enterleave").hover(function(e) { //根据事件对象的 type 判断触发了何种事件
                e.type=="mouseenter"
                ? $("#enter").text(parseInt($("#enter").text()) + 1)
                : $("#leave").text(parseInt($("#leave").text()) + 1)
            });
        });
    </script>
</body>
</html>

```

上述代码的运行效果如图 7-6 所示。

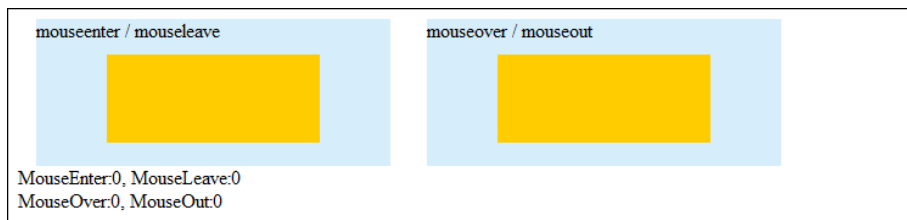


图 7-6 mouse 事件对比

上面的例子中有两个含有内部元素的 div，分别绑定了 mouseenter/mouseleave 和 mouseover/mouseout 事件。内部元素为黄颜色区域。

分别将鼠标从外部移动到黄颜色区域。注意到一共触发了一次 mouseenter 事件，如图 7-7 所示。





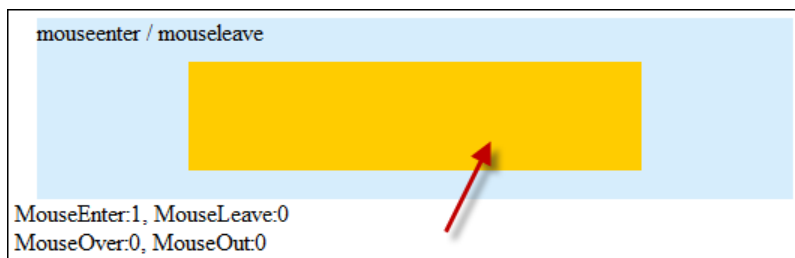


图 7-7 mouseenter 和 mouseleave 效果

但是却触发了两次 mouseover 和一次 mouseout 事件，如图 7-8 所示。

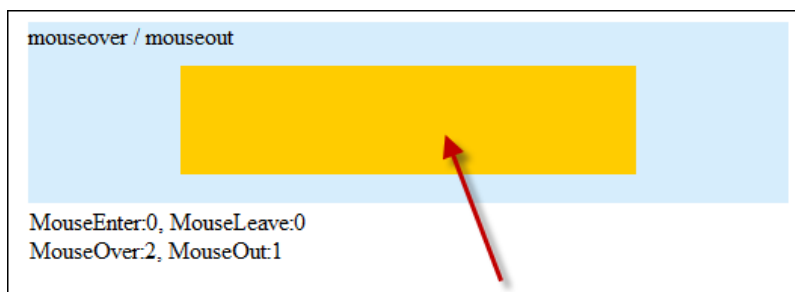


图 7-8 mouseover 和 mouseout 效果

产生这一结果的原因是鼠标在内部元素上滑动时，会触发 mouseover 和 mouseout，但是这往往不是想要的结果。mouseenter 和 mouseleave 事件解决了这一问题。

同时 jQuery 还提供了快速添加 mouseenter 和 mouseleave 事件的 hover() 函数：

```
hover( handlerIn(eventObject), handlerOut(eventObject) )
hover( handlerInOut(eventObject) )
```

当传递两个事件处理函数时，分别绑定 mouseenter 和 mouseleave 事件。

当传递一个事件处理函数时，会为 mouseenter 和 mouseleave 事件绑定同样的事件处理函数。

上面是使用 hover 替代 mouseenter() 和 mouseleave() 的例子，可以改写成：

```
$("#enterleave").hover(function(e) {
    e.type=="mouseenter"
    ? $("#enter").text(parseInt($("#enter").text()) + 1)
    : $("#leave").text(parseInt($("#leave").text()) + 1)
});
```

### 7.3.3 改进的焦点事件 focusin 和 focusout

原始的 focus 和 blur 事件分别代表获取焦点和失去焦点事件，focusin 和 focusout 的不同之处在于，如果是子元素获取了焦点或者失去焦点，同样会触发父元素的 focusin 和 focusout 事件。

假设有 HTML 片段：

```
<div id="d1" class="out">focusin / focusout <br /><input /></div>
<div id="d2" class="out">focus / blur <br /><input /></div>
```

运行效果如图 7-9 所示。





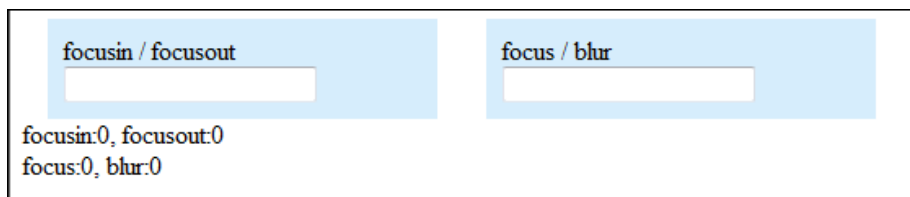


图 7-9 focusin 和 focusout

同样是两个 div，其中分别放置了一个输入框，并且分别为 div 绑定了 focusin/focusout 和 focus/blur 事件：

```
$("#d1").focusin(function(e) { //绑定 fousein 事件
    var n = parseInt($("#focusin").text()) + 1;
    $("#focusin").text(n);
}).focusout(function(e) { //绑定 fouseout 事件
    var n = parseInt($("#focusout").text()) + 1;
    $("#focusout").text(n);
});

$("#d2").focus(function() { //绑定 fouse 事件
    var n = parseInt($("#focus").text()) + 1;
    $("#focus").text(n);
}).blur(function() { //绑定 blur 事件
    var n = parseInt($("#blur").text()) + 1;
    $("#blur").text(n);
});
```

分别让两个输入框获取和失去焦点，发现会触发父类元素的 focusin 和 focusout 事件，但是不会触发父类元素的 focus 和 blur 事件，如图 7-10 所示。

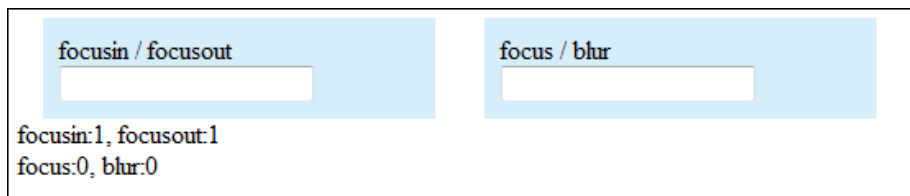


图 7-10 focusin 和 focusout 事件效果

## 7.4 小结

本章全面介绍了脚本开发中的事件和事件对象，相对于传统的做法，使用 jQuery 的各种事件函数能够简化事件开发，轻松实现跨浏览器统一。减轻工作量是好事，但是在享受 jQuery 提供的便利时，更应该了解 jQuery 内部是如何解决这些问题的。

所以本章节首先介绍了标准的 DOM 事件模型，以及事件流、事件处理函数和事件对象。通过介绍 DOM 事件模型及目前各个浏览器在处理事件上的差异，更能够体会到目前脚本开发中面临的众多不兼容问题。其中也介绍了如何解决这类问题，比如如何在各个浏览器中实现添加多事件处理函数。

本章接下来全面介绍了 jQuery 的事件和事件对象，以及 jQuery 内部特殊的事件函数。熟练地使用 jQuery 中的事件函数和事件对象，将是以后脚本开发的基础。





## 第 8 章



# 使用 AJAX 增加用户体验

AJAX 让用户页面丰富起来，增强了用户体验。使用 AJAX 是所有 Web 开发的必修课。虽然 AJAX 技术并不复杂，但实现方式还是会因为每个开发人员的方法不同而有所差异。jQuery 提供了一系列 AJAX 函数来帮助开发人员统一这种差异，并且还能方便地解决 AJAX 跨域这种技术难题。

## 8.1 原始 AJAX 与 jQuery 中的 AJAX

本节通过介绍原始的 AJAX 实现及简单 jQuery 实现，快速地领略 jQuery 的魅力。

### 8.1.1 原始 AJAX 应用举例

下面是一个原始的调用 AJAX 的例子：

```
【代码路径：jQueryStorm.Web/chapter8/Demo-OldAJAX.htm】
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>原始 AJAX 编程示例</title>
</head>
<body>
    <button id="btnAjaxOld">原始 AJAX 调用</button><br />
    <br />
    <div id="divResult"></div>
    <script type="text/javascript">
        //跨浏览器获取 XMLHttpRequest 对象
        function AjaxXmlHttpRequest() {
            var xmlhttp;
            try {
                // Firefox, Opera 8.0+, Safari
                xmlhttp = new XMLHttpRequest();
            }
            catch (e) {

                // Internet Explorer
                try {
                    xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
                }
                catch (e) {
                    try {
                        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
                    }
                    catch (e) {
                        alert("您的浏览器不支持 AJAX！");
                        return false;
                    }
                }
            }
        }
        return xmlhttp;
    }
    //使用 XMLHttpRequest 对象发送请求
    var xhr = new AjaxXmlHttpRequest();
    document.getElementById("btnAjaxOld").onclick = function () {
        var xhr = new AjaxXmlHttpRequest();
        xhr.onreadystatechange = function () {
            if (xhr.readyState == 4) {
                document.getElementById("divResult").innerHTML = xhr.responseText;
            }
        }
        xhr.open("GET", "../data/AjaxGetCityInfo.aspx?resultType=html", true);
        xhr.send(null);
    }
</script>
</body>
</html>
```





上面的实例中，“data/AjaxGetCityInfo.aspx?resultType=html”地址会返回一段 HTML 代码。使用原始 AJAX 需要做较多的事情，比如创建 XmlHttpRequest 对象、判断请求状态、编写回调函数等。

接下来看 jQuery 实现相同的功能有多简单。

### 8.1.2 jQuery 中的 AJAX 快餐

用 jQuery 的 Load() 方法，只需要一句话即可实现上面例子中的效果：

```
$("#divResult").load("data/AjaxGetCityInfo.aspx", { "resultType": "html" });
```

很多 Web 开发人员为了最高的代码灵活性，使用原始 AJAX。但是因为没有统一的封装，会发现在系统中到处分布着创建 XmlHttpRequest() 方法的函数，或者某些 AJAX 程序代码的逻辑性和结构性很差很难看懂。

虽然可以将通用的方法放到一个 js 文件中，建立自己的公共类库，然后告诉大家“嘿伙伴们，都来用这个 js 中的方法”，但是在某些时候有些新来的开发人员并不知道有这个 js 文件的存在。而且在构建自己的脚本类库过程中，肯定会遇到很多预料不到的问题，并且很难比 jQuery 做得更好！

所以请放弃制造轮子计划。使用 jQuery 的 AJAX 方法，将使一切变得更加简单。下面是使用 jQuery 实现 AJAX 的完整代码。

【代码路径：jQueryStorm.Web/chapter8/Demo-NewAjax.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>jQuery 中的 Ajax</title>
<script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
<script type="text/javascript">
    $(function () {
        $("#btnAjaxJquery").click(function (event) {
            //使用 jQuery 中的 load() 函数完成 AJAX。
            $("#divResult").load("../data/AjaxGetCityInfo.aspx", { "resultType": "html" });
        });
    })
</script>
</head>
<body>
<button id="btnAjaxJquery">使用 jQuery 的 load 方法</button>
<br />
<div id="divResult"></div>
</body>
</html>
```

通过对比原始的 AJAX 方法和 jQuery 中的 AJAX 方法，可以体会到使用 jQuery 带给开发人员的便捷。在初步领略 jQuery 的 AJAX 以后，下面开始详细地介绍 jQuery 中与 AJAX 相关的函数。

## 8.2 使用 jQuery 的 AJAX 函数进行页面交互

jQuery 中的 AJAX 函数分为如下几类，具体如表 8-1 所示。



表 8-1 AJAX 子分类列表

分类名称	分类说明
Global Ajax Event Handlers	全局的 AJAX 事件处理函数
Helper Functions	和 AJAX 相关的帮助方法，比如 serialize()函数
Low-Level Interface	底层的 AJAX()函数和 ajaxSetup()函数分类
Shorthand Methods	快捷方法分类，包括 load，get，post 等

jQuery 提供了几个用于发送 AJAX 请求的函数。其中最核心最复杂的是 jQuery.ajax ( options )，所有其他的 AJAX 函数都是它的一个简化调用。当想要完全控制 AJAX 时可以使用， 否则还是使用快捷方法如 get()、post()、load()等更加方便。

所以首先来介绍“Shorthand Methods”快捷方法分类中的函数。

### 8.2.1 AJAX 快捷函数

AJAX 快捷函数分类中，包含的函数如表 8-2 所示。

表 8-2 AJAX-> Shorthand Methods 分类函数列表

函数名称	函数说明	函数举例
jQuery.get()	使用 GET 请求获取数据	请求 test.aspx 网页，忽略返回值： \$.get("test.aspx");
jQuerygetJSON()	使用 GET 请求获取 JSON 数据	从 test.js 载入 JSON 数据并显示 JSON 数据中一个 name 字段数据： \$.getJSON("test.js", function(json){ alert("JSON Data: " + json.users[3].name); });
jQuery.getScript()	使用 GET 请求获取脚本文件并执行	加载并执行 test.js： \$.getScript("test.js");
.load()	从服务器端获取 HTML 内容并添加到匹配元素上	加载 test.html 文件内容： \$("#result").load("test.html");
jQuery.post()	使用 POST 请求获取数据	发送带参数的请求给 test.aspx 页面： \$.post("test.aspx", { name: "John", time: "2pm" } );

接下来分别讲解这几个函数。

#### 1. Load()函数

Load()函数签名列表，如表 8-3 所示。

表 8-3 Load()函数签名列表

函数签名	返回值	加入版本	参数说明
.load( url, [ data ], [ complete(responseText, textStatus, XMLHttpRequest) ] )	jQuery	1.0	url: AJAX 请求的页面地址 data: string 或者名/值对的对象。可选参数，如果传递则使用 POST 方法。 complete(responseText, textStatus, XMLHttpRequest): 回调函数，接受三个参数。其中 responseText 是返回的内容, textStatus 是状态码, XMLHttpRequest 是用于发送请求的 XmlHttpRequest 对象

load 是最简单的 AJAX 函数，注意在事件函数中也有一个 load()函数，jQuery 内部通过



传递的参数类型确定到底使用哪一个重载。load()函数最简单的用法是传递一个 url 参数：

```
//发送 get 请求
$("#divResult").load("../data/AjaxGetMethod.aspx?param=btnAjaxGet_click" + "&timestamp=" + (new Date()).getTime());
```

此时使用 GET 方式，从指定的 url 中获取返回值，并将返回值填充到 divResult 元素中。

## 注意

要时刻注意浏览器缓存，当使用 get 方式时要添加时间戳参数 (new Date()).getTime() 来保证每次发送的 URL 不同，可以避免浏览器缓存。

load()函数的第二个参数是可选参数，如果传递了则使用 Post 方式发送数据：

```
//发送 post 请求
$("#divResult").load("../data/AjaxGetMethod.aspx", { "param": "btnAjaxPost_click" });
```

回调函数作为可选参数，可以用来处理返回值：

```
//发送 Post 请求， 返回后执行回调函数
$("#divResult").load("../data/AjaxGetMethod.aspx",
{ "param": "btnAjaxCallBack_click" },
function(responseText, textStatus, XMLHttpRequest) {
    responseText = " Add in the CallBack Function! <br/>" + responseText
    $(this).html(responseText);
});
```

load()函数虽然简单，使用却具有局限性：

1. 它主要用于直接返回 HTML 的 AJAX 接口。
2. load()是一个 jQuery 对象方法，需要在 jQuery 对象调用，并且会将返回的 HTML 加载到 JQuery 对象中，即使设置了回调函数也还是会加载。

使用 load()函数的 url 参数，还可以使用“空格+选择器表达式”的方式筛选内容，比如：

```
$('#result').load('ajax/test.html #container');
```

此实例会从返回的 test.html 内容中，选择 ID 为 container 的元素，并把此元素插入到 result 元素中。

## 注意

当在 url 参数后面添加空格但没有填写选择器，则会出现“无法识别符号”的错误，请求还是能正常发送，但是无法加载 HTML 到 DOM。删除后问题解决。

### 2. get 和 post 函数

get 和 post 函数用来发送 get 和 post 请求，下面是 get 和 post 函数的重载列表，如表 8-4 所示。

表 8-4 get 和 post 函数重载列表

函数签名	返回值	加入版本	参数说明
jQuery.get( url, [ data ], [ callback(data, textStatus, XMLHttpRequest) ] )	XMLHttpRequest	1.0	url: AJAX 请求的页面地址。 data: string 或者名/值对的对象。可选参数，如果传递



(续表)

函数签名	返回值	加入版本	参数说明
XMLHttpRequest) ] , [ dataType ] )			则使用 post 方法。 complete(responseText, textStatus, XMLHttpRequest): 回调函数, responseText 是返回的内容, textStatus 是状态码, XMLHttpRequest 是用于发送请求的 XmlHttpRequest 对象 datatype: 可选参数, 返回的内容类型
jQuery.post( url, [ data ], [ success(data, textStatus, XMLHttpRequest) ] , [ dataType ] )	XMLHttpRequest	1.0	url: AJAX 请求的页面地址。 data: string 或者名/值对的对象。可选参数, 如果传递则使用 post 方法。 success (responseText, textStatus, XMLHttpRequest): 回调函数, responseText 是返回的内容, textStatus 是状态码, XMLHttpRequest 是用于发送请求的 XmlHttpRequest 对象 datatype: 可选参数, 返回的内容类型

因为 get 和 post 方法的使用十分相似, 所以放在一起讲解。

get 函数其实是对如下 AJAX 函数的封装调用。

```
$.ajax({
    url: url,
    data: data,
    success: success,
    dataType: dataType
});
```

post 函数其实是对以下 AJAX 函数的封装调用。

```
$.ajax({
    type: 'POST',
    url: url,
    data: data,
    success: success
    dataType: dataType
});
```

get 函数和 post 函数的区别就是 type 不同, get 函数使用了默认的“GET”方式发送数据, 而 post 函数使用了“POST”方式向服务器端发送数据。

在 jQuery1.4 中, 回调函数的签名有了变化:

```
callback(data, textStatus, XMLHttpRequest)
```

新增了 XMLHttpRequest 参数。

get 函数和 post 函数的用法十分相似, 比如发送请求到一个页面:

```
$.get("test.aspx");
$.post("test.aspx ");
```

发送包含数据的请求:

```
$.get("test.aspx ", { name: "John", time: "2pm" });
$.post("test.aspx ", { name: "John", time: "2pm" })
```

发送表单数据给服务器端, 并使用回调函数获取返回值:



```
$("#btnAjaxGet").click(function(e) {  
    //发送 get 请求  
    $.get("../data/AjaxGetMethod.aspx?timestamp=" + (new Date()).getTime()  
        , $("#testForm").serialize()  
        , function(data, textStatus, XMLHttpRequest) { console(data); });  
});  
  
$("#btnAjaxPost").click(function(e) {  
    //发送 post 请求  
    $.post("../data/AjaxGetMethod.aspx"  
        , $("#testForm").serialize()  
        , function(data, textStatus, XMLHttpRequest) { console(data); });  
});
```

使用 get 函数和 post 函数，有以下注意事项：

- ❑ get 函数发送 GET 类型的 HTTP 请求，有可能引起页面缓存，添加一个每次都变化的时间戳变量可以清除缓存。也可以通过 ajaxSetup()函数的 cache 和 ifModified 属性设置全局缓存模式。
- ❑ post 函数发送的 POST 类型的 HTTP 请求永远不会被缓存，此时设置 ajaxSetup()函数的 cache 和 ifModified 属性不起作用。
- ❑ get 函数可以同时 url 参数和 data 参数中传递数据，都会发送给服务器端。

### 3. getJSON()函数

getJSON()函数签名列表，如表 8-5 所示。

表 8-5 getJSON()函数签名列表

函数签名	返回值	加入版本	参数说明
jQuery.getJSON( url, [ data ], [ callback(data, textStatus) ] )	XMLHttpRequest	1.0	url: AJAX 请求的页面地址 data: string 或者名/值对的对象。可选参数，如果传递则使用 post 方法 callback (responseText, textStatus): 回调函数，responseText 是返回的内容，textStatus 是状态码

getJSON()函数是对 AJAX 函数的如下封装。

```
$.ajax({  
    url: url,  
    dataType: 'json',  
    data: data,  
    success: success  
});
```

getJSON()函数使用 GET 方式从服务器端获取 JSON 格式的数据。一旦将 dataType 属性设置为 JSON，那么传递给回调函数的 data 数据将是对象：

```
$.getJSON("test.aspx", function(data) {  
    alert(data.name);  
});
```

如果返回的数据不是 JSON 格式，那么会导致失败并且不会引发提示。所以应尽量避免 JSON 格式引起的错误。

如果 url 是另一个域的地址，则会触发跨域行为，getJSON()函数会将 AJAX 函数的





dataTType 设置为 jsonp。有关 jsonp 跨域 AJAX 将在后面做详细讲解。

下面的例子演示从 flick 的 API 中获取最新的四个图片：

```

<!DOCTYPE html>
<html>
<head>
  <style>img{ height: 100px; float: left; }</style>
  <script src="../js/jquery-1.4.2.js" type="text/javascript"></script>
</head>
<body>
  <div id="images">
</div>
<script>
//使用 getJSON()方法
$.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?tags=cat&tagmode=any&format=json&jsoncall
back=?",
    function(data){
      $.each(data.items, function(i, item){
        $("<img/>").attr("src", item.media.m).appendTo("#images");
        if ( i == 3 ) return false;
      });
    });
</script>
</body>
</html>

```

### 4. getScript()函数

getScript()函数签名列表，如表 8-6 所示。

表 8-6 getScript()函数签名列表

函数签名	返回值	加入版本	参数说明
jQuery.getScript ( url , [ data ], [success (data , textStatus) ] )	XMLHttpRequest	1.0	url: AJAX 请求的页面地址 data: string 或者名/值对的对象。可选参数，如果传递则使用 post 方法 success (responseText, textStatus): 回调函数, responseText 是返回的内容, textStatus 是状态码

getScript()函数是对 AJAX 函数的如下封装：

```

$.ajax({
  url: url,
  dataType: 'script',
  success: success
});

```

getScript()函数使用 get 方式获取 url 返回的数据，并且把数据当成是 JavaScript 脚本，立刻执行。

getScript()函数的最大好处就是可以跨域加载并执行脚本：

```
$.getScript("http://OtherWebsite/test.js");
```

其实这也是 jsonp 内的主要实现原理。

在 getScript()函数的内部实现上，首先会建立一个 script 元素，并将此元素放在 head 元素中：





```
var head = document.getElementsByTagName("head")[0];
var script = document.createElement("script");
script.src = s.url;
head.appendChild(script);
```

使用这种方式实现了跨域加载脚本。

当脚本加载后会执行，执行完毕后再从 head 中删除，下面是 jQuery 内部的实现：

```
//处理脚本加载
if ( !jsonp ) {
    var done = false;

    //为所有浏览器添加 onload 事件处理函数
    script.onload = script.onreadystatechange = function() {
        if ( !done && (!this.readyState ||
            this.readyState == "loaded" || this.readyState == "complete") ) {
            done = true;
            success();
            complete();
            //处理 IE 下的内存泄漏问题
            script.onload = script.onreadystatechange = null;
            head.removeChild( script );
        }
    };
}
```

## 注意

如果使用 getScript()获取跨域的文件，那么在回调函数中的 data 和 textStatus 为 undefined。

通过 getJSON()和 getScript()可以发现，这两个函数其实都是对 ajax()函数的 dataType 属性的不同属性值的封装。所以可见 dataType 属性的重要性。如果需要甚至可以封装自己的 AJAX 快捷函数。既然所有的快捷函数都是对 ajax()函数的封装，那么下面就来系统地学习 ajax()函数。

### 8.2.2 底层函数 ajax()和 ajaxSetup()

在 jQuery 1.4 版本中，将这两个函数分成了一类，放在了 Ajax->Low-Level Interface 分类下。

下面是 ajax()函数和 ajaxSetup()函数的重载列表，如表 8-7 所示。

表 8-7 ajax()函数和 ajaxSetup()函数重载列表

函数签名	返回值	加入版本	参数说明
jQuery.ajax(options)	XMLHttpRequest	1.0	options: 包含很多属性值的对象，用于设置 AJAX 请求
jQuery.ajaxSetup(options)	void	1.1	options: 包含很多属性值的对象，用于设置 AJAX 请求

ajax()函数是 jQuery 实现 AJAX 的核心函数。其关键点在于此函数的唯一参数 options。options 包含了很多的属性用于设置 AJAX 请求的各种行为：

ajaxSetup()函数的唯一参数是和 ajax()函数的参数相同的，ajax()函数使用时可以不传递 options，则各种属性会使用默认值。ajaxSetup()可以设置全局的 options 对象的默认值。下面是 options 对象的属性列表，如表 8-8 所示。



## 第 8 章 使用 AJAX 增加用户体验

表 8-8 options 对象属性列表

属性名称	属性类型	属性说明
async	Boolean	(默认 true) 默认设置下, 所有请求均为异步请求。如果需要发送同步请求, 请将此选项设置为 false。注意, 同步请求将锁住浏览器, 用户其他操作必须等待请求完成才可以执行
beforeSend(XHR)	Function	发送请求前可修改 XMLHttpRequest 对象的函数, 如添加自定义 HTTP 头。XMLHttpRequest 对象是唯一的参数。这是一个 AJAX 事件。如果返回 false 可以取消本次 AJAX 请求
cache	Boolean	(默认 true, dataType 为 script 和 jsonp 时默认为 false) jQuery 1.2 新功能, 设置为 false 将不缓存此页面
complete(XHR, TS)	Function	请求完成后回调函数 (请求成功或失败之后均调用)。参数: XMLHttpRequest 对象和一个描述成功请求类型的字符串。AJAX 事件
contentType	String	(默认: "application/x-www-form-urlencoded") 发送信息至服务器时内容编码类型。默认值适合大多数情况。如果你明确地传递了一个 content-type 给 \$.ajax(), 那么它必定会发送给服务器 (即使没有数据要发送)
context	Object	这个对象用于设置 AJAX 相关回调函数的上下文。也就是说, 让回调函数内 this 指向这个对象 (如果不设定这个参数, 那么 this 就指向调用本次 AJAX 请求时传递的 options 参数)。比如指定一个 DOM 元素作为 context 参数, 这样就设置了 success 回调函数的上下文为这个 DOM 元素。就像这样: <pre>\$.ajax({ url: "test.html", context: document.body, success: function(){     \$(this).addClass("done"); }});</pre>
data	Object, String	发送到服务器的数据。将自动转换为请求字符串格式。get 请求中将附加在 URL 后。查看 processData 选项说明已禁止此自动转换。必须为 Key/Value 格式。如果为数组, jQuery 将自动为不同值对应同一个名称。如 {foo:["bar1", "bar2"]} 转换为 '&foo=bar1&foo=bar2'
dataFilter	Function	给 AJAX 返回的原始数据进行预处理的函数。提供 data 和 type 两个参数: data 是 AJAX 返回的原始数据, type 是调用 jQuery.ajax 时提供的 dataType 参数。函数返回的值将由 jQuery 进一步处理
dataType	String	预期服务器返回的数据类型。如果不指定, jQuery 将自动根据 HTTP 包 MIME 信息来智能判断, 比如 XML MIME 类型就被识别为 XML。在 jQuery 1.4 中, JSON 就会生成一个 JavaScript 对象, 而 script 则会执行这个脚本。随后服务器端返回的数据会根据这个值解析后, 传递给回调函数。可用值: “xml”: 返回 XML 文档, 可用 jQuery 处理。 “html”: 返回纯文本 HTML 信息; 包含的 script 标签会在插入 dom 时执行。 “script”: 返回纯文本 JavaScript 代码。不会自动缓存结果。除非设置了 “cache” 参数。 “注意: ”在远程请求时 (不在同一个域下), 所有 POST 请求都将转为 get 请求。(因为将使用 DOM 的 script 标签来加载) “json”: 返回 JSON 数据。 “jsonp”: JSONP 格式。使用 JSONP 形式调用函数时, 如 “myurl?callback=?” jQuery 将自动替换 ? 为正确的函数名, 以执行回调函数。 “text”: 返回纯文本字符串
error	Function	(默认: 自动判断 (XML 或 HTML)) 请求失败时调用此函数。有以下三个参数: XMLHttpRequest 对象、错误信息、(可选) 捕获的异常对象。如果发生了错误, 错误信息 (第二个参数) 除了得到 null 之外, 还可能是 “timeout”, “error”, “notmodified” 和 “parsererror”。AJAX 事件。 <pre>function (XMLHttpRequest, textStatus, errorThrown) {</pre>



(续表)

属性名称	属性类型	属性说明
		<pre>//通常 textStatus 和 errorThrown 之中只有一个会包含信息 this; //调用本次 AJAX 请求时传递的 options 参数 }</pre>
global	Boolean	(默认 true) 是否触发全局 AJAX 事件。设置为 false 将不会触发全局 AJAX 事件, 如 ajaxStart 或 ajaxStop 可用于控制不同的 AJAX 事件
ifModified	Boolean	(默认: false) 仅在服务器数据改变时获取新数据。使用 HTTP 包 Last-Modified 头信息判断。在 jQuery 1.4 中, 它也会检查服务器指定的 etag 来确定数据没有被修改过
jsonp	String	在一个 JSONP 请求中重写回调函数的名字。这个值用来替代在"callback=?"这种 get 或 post 请求中 URL 参数里的"callback"部分, 比如 {jsonp:'onJsonPLoad'} 会导致将"onJsonPLoad=?" 传给服务器
jsonpCallback	String	为 JSONP 请求指定一个回调函数名。这个值将用来取代 jQuery 自动生成的随机函数名。这主要用来让 jQuery 生成独特的函数名, 这样管理请求更容易, 也能方便地提供回调函数和错误处理。也可以在想让浏览器缓存 get 请求的时候, 指定这个回调函数名
password	String	用于响应 HTTP 访问认证请求的密码
processData	Boolean	(默认 true) 默认情况下, 通过 data 选项传递进来的数据, 如果是一个对象 (技术上讲只要不是字符串), 都会处理转化成 一个查询字符串, 以配合默认内容类型 "application/x-www-form-urlencoded"。如果要发送 DOM 树信息或其他不希望转换的信息, 请设置为 false
scriptCharset	String	只有当请求时 dataType 为"jsonp"或"script", 并且 type 是"GET"才会用于强制修改 charset。通常只在本地和远程的内容编码不同时使用
success	Function	<p>请求成功后的回调函数。参数: 由服务器返回, 并根据 dataType 参数进行处理后的数据; 描述状态的字符串。AJAX 事件。</p> <pre>function (data, textStatus) {     //data 可能是 xmlDoc, jsonObj, html, text 等...     this; //调用本次 AJAX 请求时传递的 options 参数 }</pre>
traditional	Boolean	如果想要用传统的方式序列化数据, 那么就设置为 true。请参考工具分类下面的 jQuery.param 方法
timeout	Number	设置请求超时时间 (毫秒)。此设置将覆盖全局设置
type	String	(默认 "GET") 请求方式 ("POST" 或 "GET"), 默认为 "GET"。注意, 其他 HTTP 请求方法, 如 PUT 和 DELETE 也可以使用, 但仅部分浏览器支持
url	String	(默认当前页地址) 发送请求的地址
username	String	用于响应 HTTP 访问认证请求的用户名
xhr	Function	需要返回一个 XMLHttpRequest 对象。默认在 IE 下是 ActiveXObject 而其他情况下是 XMLHttpRequest。用于重写或者提供一个增强的 XMLHttpRequest 对象。这个参数在 jQuery 1.3 以前不可用

看着这些属性有没有望而生畏的感觉? 正因为 options 属性值太多, 所以 jQuery 才提供了简便的 get()函数、post()函数等。毕竟选中 jQuery 就是看中了它的轻便简单。

但是详细地了解 ajax()函数和 ajax()函数的 options 参数, 对于底层的开发人员和高级应用开发人员来说还是很有必要的。

ajax()函数可以不带参数调用:

```
$.ajax();
```



此时将没有任何效果，并且使用 options 参数的所有默认值。上面的例子中会获取当前页面的内容，如果想要实现功能，需要首先添加回调函数。

### 1. 回调函数

options 参数中，有如下用于设置回调函数的属性。

- ❑ beforeSend 在发送请求之前调用，并且传入一个 XMLHttpRequest 作为参数。
- ❑ error 在请求出错时调用。传入 XMLHttpRequest 对象，描述错误类型的字符串及一个异常对象（如果有的话）。
- ❑ dataFilter 在请求成功之后调用。传入返回的数据及“dataType”参数的值。并且必须返回新的数据（可能是处理过的）传递给 success 回调函数。
- ❑ success 当请求之后调用。传入返回后的数据，以及包含成功代码的字符串。
- ❑ complete 当请求完成之后调用这个函数，无论成功或失败。传入 XMLHttpRequest 对象，以及一个包含成功或错误代码的字符串。

开发人员可以充分地发挥想象力，实现各种引用。比如数据过滤、同一添加验证信息等。最常用的还是 success() 函数：

```
$.ajax({  
  url: 'ajax/test.html',  
  success: function(data) {  
    $('result').html(data);  
    alert('Load was performed.');  }  
});
```

上面的例子其实可以使用 get() 函数替代。

### 2. 数据类型

ajax() 函数依赖服务器提供的信息来处理返回的数据。如果服务器报告说返回的数据是 XML，那么返回的结果就可以用普通的 XML 方法或者 jQuery 的选择器来遍历。如果是其他类型，比如 HTML，则数据就以文本形式来对待。

通过 dataType 选项还可以指定其他不同数据的处理方式。除了单纯的 XML，还可以指定 HTML、JSON、JSONP、script 或者 text。

其中，text 和 XML 类型返回的数据不会经过处理。数据仅仅简单地将 XMLHttpRequest 的 responseText 或 responseHTML 属性传递给 success 回调函数。

## 注意

必须确保网页服务器报告的 MIME 类型与选择的 dataType 所匹配。比如是 XML 时，服务器端就必须声明 text/xml 或者 application/xml 来获得一致的结果。

如果指定为 HTML 类型，任何内嵌的 JavaScript 都会在 HTML 作为一个字符串返回之前执行。类似地，指定 script 类型的话，也会先执行服务器端生成 JavaScript，然后再把脚本作为一个文本数据返回。

如果指定为 JSON 类型，则会把获取到的数据作为一个 JavaScript 对象来解析，并且把构建好的对象作为结果返回。为了实现这个目的，首先尝试使用 JSON.parse()。如果浏览器不支持，则使用一个函数来构建。JSON 数据是一种能很方便通过 JavaScript 解析的结构化





数据。如果获取的数据文件存放在远程服务器上（域名不同，也就是跨域获取数据），则需要使用 JSONP 类型。使用这种类型的话，会创建一个查询字符串参数 `callback=?`，这个参数会加在请求的 URL 后面。服务器端应当在 JSON 数据前加上回调函数名，以便完成一个有效的 JSONP 请求。如果要指定回调函数的参数名来取代默认的 `callback`，可以通过设置 `$.ajax()` 的 JSONP 参数。

## 注意

JSONP 是 JSON 格式的扩展。它要求一些服务器端的代码来检测并处理查询字符串参数。更多关于 JSONP 的内容请参见下面 JSONP 章节。

如果指定了 `script` 或者 JSONP 类型，那么当从服务器接收到数据时，实际上是用 `<script>` 标签而不是 `XMLHttpRequest` 对象。这种情况下，`$.ajax()` 不再返回一个 `XMLHttpRequest` 对象，并且也不会传递事件处理函数，比如 `beforeSend`。

### 3. 发送数据到服务器

默认情况下，AJAX 请求使用 `get` 方法。如果要使用 `post` 方法，可以设定 `type` 参数值。这个选项也会影响 `data` 选项中的内容如何发送到服务器。

`data` 选项既可以包含一个查询字符串，比如 `key1=value1&key2=value2`，也可以是一个映射，比如 `{key1: 'value1', key2: 'value2'}`。如果使用了后者的形式，则数据再发送会被转换成查询字符串。这个处理过程也可以通过设置 `processData` 选项为 `false` 来回避。如果希望发送一个 XML 对象给服务器时，这种处理可能并不合适，并且在这种情况下，也应当改变 `contentType` 选项的值，用其他合适的 MIME 类型取代默认的 `application/x-www-form-urlencoded`。

### 4. 高级选项

`global` 选项用于阻止响应注册的回调函数，比如 `ajaxSend`，或者 `ajaxError`，以及类似的方法。这在有些时候很有用，比如发送的请求非常频繁且简短时，就可以在 `ajaxSend` 里禁用这个选项。更多关于这些选项的详细信息，请参阅下面的内容。

如果服务器需要 HTTP 认证，可以使用用户名和密码通过 `username` 和 `password` 选项来设置。

AJAX 请求是限时的，所以错误警告被捕获并处理后，可以用来提升用户体验。请求超时 `options` 选项的 `timeout` 属性控制。通常保留其默认值，或通过 `jQuery.ajaxSetup` 来全局设定，很少为特定的请求重新设置 `timeout` 选项。

默认情况下，请求总会被发出去，但浏览器有可能从它的缓存中调取数据。要禁止使用缓存的结果，可以设置 `cache` 参数为 `false`。如果希望判断数据自从上次请求后没有更改过就报告出错的话，可以设置 `ifModified` 为 `true`。

`scriptCharset` 允许给 `<script>` 标签的请求设定一个特定的字符集，用于 `script` 或者 `jsonp` 类似的数据。当脚本和页面字符集不同时，这点特别好用。

AJAX 的第一个字母是 `asynchronous` 的开头字母，这意味着所有的操作都是并行的，完成的顺序没有前后关系。`options` 选项的 `async` 属性总是设置成 `true`，这标志着在请求开始后，其他代码依然能够执行。如果设置为 `false` 则可以执行同步的 AJAX 请求。





ajax()函数返回它创建的 XMLHttpRequest 对象。通常 jQuery 只在内部处理并创建这个对象，但用户也可以通过 xhr 选项传递一个自己创建的 xhr 对象。虽然返回的对象通常已经被丢弃了，但依然提供一个底层接口来观察和操控请求。比如说，调用对象上的.abort()可以在请求完成前挂起请求。

下面通过一些示例来演示 ajax()函数的使用：

(1) 加载并执行一个 JS 文件：

```
$.ajax({
  type: "GET",
  url: "test.js",
  dataType: "script"
});
```

(2) 保存数据到服务器，成功时显示信息如下：

```
$.ajax({
  type: "POST",
  url: "test.aspx",
  data: "name=John&location=Boston",
  success: function(msg){
    alert( "Data Saved: " + msg );
  }
});
```

(3) 装入一个 HTML 网页最新版本：

```
$.ajax({
  url: "test.html",
  cache: false,
  success: function(html){
    $("#results").append(html);
  }
});
```

(4) 同步加载数据。发送请求时锁住浏览器。需要锁定用户交互操作时使用同步方式：

```
var html = $.ajax({
  url: "test.aspx",
  async: false
}).responseText;
```

(5) 发送 XML 数据至服务器。设置 processData 选项为 false，防止自动转换数据格式：

```
var xmlDocument = [create xml document];
$.ajax({
  url: "page.php",
  processData: false,
  data: xmlDocument,
  success: handleResponse
});
```

### 8.2.3 AJAX 帮助函数

在 AJAX-> Helper Functions 分类中，保存了使用 AJAX 时常用到的工具类函数，如表 8-9 所示。



表 8-9 Helper Functions 分类函数签名列表

函数签名	返回值	加入版本	参数说明
jQuery.param( obj )	string	1.2	obj: 要被序列化的数组或对象
jQuery.param( obj, traditional )	string	1.4	obj: 要被序列化的数组或对象 traditional: 布尔值, 是否使用传统的方式浅层序列化
.serialize()	string	1.0	将成功的表单元素序列化成 url 查询参数字符串
.serializeArray()	Array	1.2	将表单元素序列化成一个个名/值对的集合

### 1. param()函数

param()函数是.serialize()的核心方法, 主要作用就是将一个对象或者数组, 序列化成字符串:

```
var obj = { name: "zhangziqu", age: "999" };  
console.log(jQuery.param(obj)); //输出 name=zhangziqu&age=999
```

param 函数主要是用来序列化名/值对的字符串, 所以主要是序列化对象。

jQuery 1.4 版本的 param()函数在序列化复杂对象时, 有很大的改变:

```
//版本号 1.3.2 及以下:  
$.param({ a: [2,3,4] }) // "a=2&a=3&a=4"  
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a=[object+Object]&d=3&d=4&d=[object+Object]"  
//版本号 1.4 及以上  
$.param({ a: [2,3,4] }) // "a[]=2&a[]=3&a[]=4"  
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a[b]=1&a[c]=2&d[]=3&d[]=4&d[2][e]=5"
```

在 jQuery 1.4 中, .param()会深度递归一个对象来满足现在脚本语言和框架, 比如 PHP、Ruby on Rails 等。如同上面的 “{ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }” 这个对象。这个对象具有两个属性 a 和 b, 其中 a 和 b 又分别是对象。在 1.3.2 及以前版本的 jQuery 中是不支持深层次序列化的, 但是在 1.4 版本中已经开始支持了。可以为 param()函数传递第二个参数来确定是否使用深层次序列化, 默认为 false, 即 “不使用浅层序列化”。需要注意, 使用深层次序列化后的字符串是被编码后的, 比如:

```
$.param({ a: { b: 1, c: 2 }, d: [3, 4, { e: 5 }] })  
//输出: a%5Bb%5D=1&a%5Bc%5D=2&d%5B%5D=3&d%5B%5D=4&d%5B2%5D%5Be%5D=5
```

可以使用 decodeURIComponent()函数进行解码:

```
decodeURIComponent($.param({ a: { b: 1, c: 2 }, d: [3, 4, { e: 5 }] })))  
//输出: a[b]=1&a[c]=2&d[]=3&d[]=4&d[2][e]=5
```

还可以通过设置 jQuery.ajaxSettings.traditional 属性, 全局禁用这个功能:

```
jQuery.ajaxSettings.traditional = true;
```

### 注意

因为有些框架在解析序列化数字的时候能力有限, 所以当传递一些含有嵌套对象、数组的对象作为参数时, 请务必小心。

在 jQuery 1.4 中, HTML5 的 input 元素也会被序列化。





下面演示按照 key/value 对序列化普通对象。

```
var params = { width:1680, height:1050 };
var str = jQuery.param(params);
$("#results").text(str); //输出: width=1680&height=1050
```

下面演示深层序列化和浅层序列化，以及如何解码。

```
var myObject = {
  a: {
    one: 1,
    two: 2,
    three: 3
  },
  b: [1,2,3]
};
//使用 param()函数序列化对象
var recursiveEncoded = $.param(myObject);
//对序列化以后的字符串解码
var recursiveDecoded = decodeURIComponent($.param(myObject));
console(recursiveEncoded);
//输出结果为:
a%5Bone%5D=1&a%5Btwo%5D=2&a%5Bthree%5D=3&b%5B%5D=1&b%5B%5D=2&b%5B%5D=3
console(recursiveDecoded);
//输出结果为: a[one]=1&a[two]=2&a[three]=3&b[]=1&b[]=2&b[]=3
var shallowEncoded = $.param(myObject, true);
var shallowDecoded = decodeURIComponent(shallowEncoded);
console(shallowEncoded);
//输出结果为: a=%5Bobject+Object%5D&b=1&b=2&b=3
console(shallowDecoded);
//输出结果为: a=[object+Object]&b=1&b=2&b=3
```

最后还有几点需要注意，如果传递对象的某一个属性是一个函数，那么用.param()会得到这个函数的返回值，而序列化这个函数：

```
var obj = { name: "zhangziqu", age: "999", abc: function() { return "bbb"; } };
console($.param(obj)); //输出: name=zhangziqu&age=999&abc=bbb
```

## 2. serialize()函数

serialize()函数可以将表单元素编码成一个用于提交的查询字符串。

比如如果有表单元素：

```
<input type="text" name="a" value="1" id="a" />
<input type="text" name="b" value="2" id="b" />
```

序列化后的结果为：

```
$("#form").serialize(); //输出: a=1&b=2
```

serialize()函数的关键点是只序列化“有效”的表单元素。所谓有效，是指：

- ☐ 不能是 disabled。
- ☐ 选中的复选框、单选按钮或 select 元素。
- ☐ 隐藏的表单元素如果满足上面条件也是有效的表单元素。

具体有效元素的说明，可以参见 W3C 的官方文档：

<http://www.w3.org/TR/html401/interact/forms.html#h-17.13.2>。

下面的示例演示 serialize()函数在各表单元素上的效果。



【代码路径: jQueryStorm.Web/chapter8/ Demo-serialize.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>jQuery Ajax - param </title>
<script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
<script type="text/javascript">
    function console(text) {
        $("#divMsg").append("<div>" + text + "</div>");
    }
    $(function() {
        $("#btnSerialize").click(function(e) {
            console($("#form").serialize());           //序列化表单
        });
    })
</script>

</head>
<body>
<form>
<select name="single">
<option>Single</option>
<option>Single2</option>
</select>
<select name="multiple" multiple="multiple">
<option selected="selected">Multiple</option>
<option>Multiple2</option>
<option selected="selected">Multiple3</option>
</select>
<div>
<input type="text" name="a" value="1" id="a" />
<input type="text" name="b" value="2" id="b" />
</div>
<div>
<input type="checkbox" name="f" value="8" id="f" />
</div>
<input type="button" id="btnSerialize" value="serialize" />
</form>
<div id="divMsg"></div>
</body>
</html>
```

效果如图 8-1 所示。

图 8-1 serialize()函数效果

通过实例可以看出,如果某一个 select 允许选中多项,则在查询字符串中会出现同名的键值:

multiple=Multile&multiple=Multiple3

如果将这个值传递给服务器端，ASP.NET 将获取到使用 “,” 逗号拼接的多个值的字符串：

```
var a = Request["multiple "]; //a 的值为 “Multile, Multiple3”
```

另外，每个表单元素一定要有 “name” 属性才能被正确地序列化。

### 3. serializeArray()函数

serializeArray()函数也是序列化表单对象，但是返回的结果是一个对象集合，集合中的每个元素是一个对象，包含 name 属性和 value 属性。格式如下：

```
[
  {name: 'firstname', value: 'Hello'},
  {name: 'lastname', value: 'World'},
  {name: 'alias'}, // this one was empty
]
```

如果想要遍历获取到的集合，可以使用 each()函数：

```
var fields = $("#input").serializeArray();
$("#results").empty();
jQuery.each(fields, function(i, field){
  $("#results").append(field.value + " ");
});
```

## 8.2.4 AJAX 全局事件

AJAX 全局事件，是指在所有发生 AJAX 请求时都会触发的事件。

在 jQuery.ajaxSetup( options ) 中的 options 参数属性中，有一个 global 属性，控制是否引发全局事件，并且默认为 true。

jQuery 包括如下 AJAX 全局事件，如表 8-10 所示。

表 8-10 AJAX 全局事件列表

事件函数	事件说明
.ajaxComplete()	AJAX 请求完成时执行函数
.ajaxError()	AJAX 请求发生错误时执行函数
.ajaxSend()	AJAX 请求发送前执行函数
.ajaxStart()	AJAX 请求开始时执行函数
.ajaxStop()	AJAX 请求结束时执行函数
.ajaxSuccess()	AJAX 请求成功时执行函数

这些全局事件会在执行任何一个 AJAX 请求时触发。

用一个示例讲解各个事件的触发顺序：

【代码路径：jQueryStorm.Web/chapter8/ Demo-AjaxGlobalEvent.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery 全局 AJAX 事件</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
  <script type="text/javascript">
    $(document).ready(function ()
```



```

{
    //单击时执行一次 AJAX 请求
    $("#btnAjax").bind("click", function (event)
    {
        $.get("../data/AjaxGetMethod.aspx");
    })
    //添加 “ajaxComplete” 事件
    $("#divResult").ajaxComplete(function (evt, request, settings)
    { $(this).append('<div>ajaxComplete</div>'); })
    //添加 “ajaxError” 事件
    $("#divResult").ajaxError(function (evt, request, settings)
    { $(this).append('<div>ajaxError</div>'); })
    //添加 “ajaxSend” 事件
    $("#divResult").ajaxSend(function (evt, request, settings)
    { $(this).append('<div>ajaxSend</div>'); })
    //添加 “ajaxStart” 事件
    $("#divResult").ajaxStart(function () { $(this).append('<div>ajaxStart</div>'); })
    //添加 “ajaxStop” 事件
    $("#divResult").ajaxStop(function () { $(this).append('<div>ajaxStop</div>'); })
    //添加 “ajaxSuccess” 事件
    $("#divResult").ajaxSuccess(function (evt, request, settings)
    { $(this).append('<div>ajaxSuccess</div>'); })
});
</script>
</head>
<body>
    <br /><button id="btnAjax">发送 Ajax 请求</button><br/>
    <div id="divResult"></div>
</body>
</html>

```

当单击此示例的按钮时，会执行一次 AJAX 请求，页面上会输出此请求触发的各种全局事件，效果如图 8-2 所示。



图 8-2 AJAX 全局事件举例

## 8.3 跨域的 AJAX-JSONP

使用传统的 XMLHttpRequest 对象实现 AJAX 请求，无法实现跨域操作，甚至无法简单地跨子域。JSONP 可以解决 AJAX 的跨域问题。本节将全面地介绍 JSONP 及 JSONP 在 jQuery 中的实现。

### 8.3.1 什么是 JSONP

JSONP 是 JSON with Padding 的缩写，是一个非官方的协议。



JSON 已经在第 2 章“必须知道的 JavaScript 知识”中介绍过，是一种用于表示对象的数据格式。下面是一个简单的 JSON 串：

```
{ name: "zhangziqu", isGood: true }
```

可以使用这个字符串创建对象，并访问创建的属性。

```
var obj = { name: "zhangziqu", isGood: true };  
console(obj.name); //输出: "zhangziqu"  
console(obj.isGood); //输出: "true"
```

JSONP 这种格式是指将远程的数据使用 JSON 返回，但是要在外层加上“包装”：

```
callback({ name: "zhangziqu", isGood: true });
```

也就是把返回的数据包装在函数中，上面的“callback”就是由服务器端添加的。实际上拼接成了一条 JavaScript 语句，调用 callback() 函数，并且调用时传递了数据参数。

一般来说 callback() 这个函数的名称，是由客户端作为参数传递的。服务器端需要支持这个特殊的参数，并且把返回的 JSON 数据用 callback() 函数包装。

下面是支持 JSONP 调用的 ASP.NET 服务器端实例：

```
protected void Page_Load(object sender, EventArgs e)  
{  
    string callback = Request["callback"];  
    Response.Clear();  
    Response.Write(callback+"({ name: \"zhangziqu\", isGood: true })");  
    Response.End();  
}
```

JSONP 首先需要服务器端支持，上面的代码实现了服务器端支持 callback 这个参数，并且使用参数值包装返回的 JSON 数据。

客户端示例如下：

```
【代码路径： jQueryStorm.Web/chapter8/ Demo-JSONP.htm】  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
    <title>jQuery 使用 JSONP </title>  
    <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>  
    <script type="text/javascript">  
        function myJsonCallBack(data) {  
            console(data.name);  
        }  
        function console(text) {  
            $("#divMsg").append("<div>" + text + "</div>");  
        }  
        $(function() {  
            $.ajax({  
                url: "../../data/AjaxJSONP.aspx",  
                dataType: "jsonp",  
                jsonpCallback: "myJsonCallBack"  
            });  
        })  
    </script>  
</head>  
<body>  
    <div id="divMsg"></div>
```





```
</body>
</html>
```

上述代码首先创建了一个函数 `myJsonCallBack()`。接下来创建了一个 `script` 元素，用来向服务器端发送请求，并把服务器端的返回值作为语句执行。

这就是一个最基础的 JSONP 的演示。

### 8.3.2 JSONP 实现原理

JSONP 之所以可以跨域，是因为 `script` 标签可以发送跨域请求。但是 `script` 标签引用的都是外部脚本，这正是为什么服务器端不能只返回数据，而是要返回一个拼接好 JavaScript 语句的原因。

虽然服务器端返回的是语句，但是语句的生成还是要根据客户端传递的参数生成。也就是说，客户端要首先和服务端约定一个参数作为回调函数的名称，比如在 jQuery 中默认 JSONP 的回调函数参数名是 `callback`，以及传递给服务器端有一个名字是 `callback` 的参数。服务器端会根据 `callback` 的值，生成 JavaScript 语句。

当执行服务器端返回的语句时，要确定回调函数此时已经被解析，否则会找不到回调函数。使用 JSONP 实现跨域的原理仅此而已，并不复杂。

### 8.3.3 JSONP 在 jQuery 中的应用

在介绍什么是 JSONP 的例子中，并没有动态地创建 `script` 元素，这是因为 jQuery 的 AJAX 函数很多已经支持了 JSONP。

其中最底层的就是 `ajax()` 函数，其 `dataType` 属性就可以设置为 JSONP，下面是使用 `ajax()` 函数实现 JSONP 的例子：

```
$.ajax({
    url: "../data/AjaxJSONP.aspx",
    dataType: "jsonp",
    success: function(data) { console(data.name); }
});
```

如果设置了 `dataType` 为 `jsonp`，默认情况下会为 `success` 这个匿名函数创建一个随机的名字，比如“`jsoncallback123`”并且在发送请求时添加“`callback= jsoncallback123`”参数。jQuery 会动态创建一个 `script` 对象，并添加到 `head`。此时添加的 `script` 会立刻去服务器端加载脚本并执行。执行完毕后再将此 `script` 删除。

如果想改变和服务端约定好的回调函数参数名称“`callback`”，可以修改 `ajax()` 函数参数的 JSONP 属性：

```
$(function() {
    $.ajax({
        url: "../data/AjaxJSONP.aspx",
        dataType: "jsonp",
        success: function(data) { console(data.name); },
        jsonp: "myCallBack"
    });
});
```

此时回调函数就会通过“`myCallBack`”这个参数传递了，也就是说以前是：



```
data/AjaxJSONP.aspx?callback=myJsonCallBack123
```

现在变成了：

```
data/AjaxJSONP.aspx? myCallBack =myJsonCallBack123
```

使用 `success()` 函数会传递随机生成的函数名称，如果希望为回调函数指定一个名称，可以修改 `ajax()` 函数参数的 `jsonpCallback` 属性：

```
$.ajax({  
    url: "http://www.jquerystorm.com/data/AjaxJSONP.aspx",  
    dataType: "jsonp",  
    success: function(data) { console(data.name); },  
    jsonpCallback: "myJsonCallBack2"  
});
```

注意，上面的例子同时使用了 `success` 和 `jsonpCallback` 两个参数，这相当于为 `success` 中的函数起了一个名字，要保证这个名字不和页面上其他的函数冲突。如果页面上也定义了一个名字为“`myJsonCallBack2()`”的函数，那么 `success` 属性传递的函数将不起作用。

除了 `ajax()` 函数，`getScript()` 函数本身就是跨域的。

`getJSON()` 函数如果请求了跨域的 URL，也会使用 JSONP，其实现就是在调用 `ajax()` 函数时将 `dataType` 设为 JSONP。

## 8.4 小结

本章介绍了如何使用 jQuery 实现 AJAX 功能，使用起来最简单的是快捷 AJAX 事件，如 `get` 函数、`post` 函数、`getJSON()` 函数、`getScript()` 函数等。

如果求最大的灵活性，则可以使用 `ajax()` 函数。`ajax()` 函数的参数十分复杂，而功能也十分强大。

AJAX 帮助函数的使用，也会对日常的程序处理大有帮助。比如使用 `param()` 函数格式化对象，使用 `serialize()` 函数格式化表单等。

在面临 AJAX 跨域问题时，JSONP 是最好的解决方案。本章还介绍了 JSONP 及如何在 jQuery 中使用 JSONP。







## 第 9 章



# jQuery 动画——让页面动起来

丰富的页面动画效果，可以让网站更加吸引用户。使用 Flash 或者 JavaScript 都可以实现动画效果，Flash 可以实现复杂的动画，甚至用来制作游戏。但是 Flash 是二进制的，需要编译成 swf 等格式的文件并且还需要插件运行。如今 HTML 5 的动画标准、苹果舍弃 Flash 等都让 Flash 的前途更加渺茫。

使用 JavaScript 制作动画就不同了，其编写简单，可以自由地操作 DOM 元素，效率很高。各大浏览器都在不停地提高 JavaScript 引擎的性能，使代码具有很高的复用性，未来 HTML 5 将有更多的支持……

虽然使用 JavaScript 实现页面动画有这么多的好处，但是在各种脚本类库出现之前是完全不同的。不仅要编写复杂动画效果代码，而且还要兼容不同的浏览器。

jQuery 类库提供了 Effect 分类函数，用来简单快捷地实现各种动画，并且提供了 jQuery UI 类库，封装了各种常用的页面 UI 控件。本章将介绍 jQuery 的动画函数，jQuery UI 将在后面的独立章节中讲解。

## 9.1 jQuery 动画基础

jQuery 中提供了 Effects 分类（此分类不包括子分类），里面存放了所有和页面效果相关的函数。在介绍具体的动画函数之前，本节将首先通过一个例子让学习者快速入门，然后讲解一下所有的 jQuery 动画函数都要用到的回调函数和时间参数的作用，为接下来学习具体的动画函数打好基础。

首先让我们通过一个实例，快速地领略 jQuery 动画的魅力。

### 9.1.1 动画入门实例

【代码路径：jQueryStorm.Web/chapter9/ Demo-Simple.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>jQuery 动画入门</title>
  <style>
    div
    {
      background: #def3ca;
      margin: 3px;
      width: 80px;
      display: none;
      float: left;
      text-align: center;
    }
  </style>
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
</head>
<body>
  <button id="show">
    Show</button>
  <button id="hid">
    Hide</button>
  <div>
    Hello
  </div>
  <div>
    jQuery!
  </div>
  <script type="text/javascript">
    $("#show").click(function () {
      $("#div").show("fast");
    });
    $("#hid").click(function () {
      $("#div").hide(1000);
    });
  </script>
</body>
</html>
```

实例效果如图 9-1 所示。

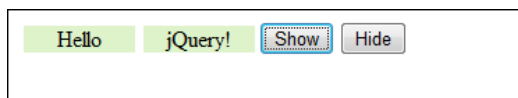


图 9-1 jQuery 动画快速入门实例



单击“Show”按钮，将会用较快速的动画效果显示“Hello jQuery”，单击“Hide”按钮时用较慢的动画效果隐藏“Hello jQuery”。

这个例子中使用了 jQuery 提供的 show()和 hide()函数，这两个函数如果不传递参数，则是单纯地显示和隐藏元素，如果传递了“fast”或者数字，则是用动画效果显示或隐藏。现在让页面动起来变得如此简单了！

### 9.1.2 jQuery 动画分类

在 jQuery API 中，按照动画实现的效果，将动画函数分类，如表 9-1 所示。

表 9-1 jQuery 动画函数分类

函数分类	函数说明
基础动画函数	使用时最简单的动画函数，包括 show()、hide()和 toggle()函数
自定义动画函数	包括队列相关函数、自定义动画函数 animate()、全局属性等
渐变动画函数	实现淡入淡出的效果
滑动动画函数	实现滑动渐变效果

在后面的内容中将分别讲解这些分类中的函数。

### 9.1.3 jQuery 动画实验室

jQuery 动画实验室可以方便地练习 jQuery 的各种动画函数，如图 9-2 所示是实验室的效果。在左侧的输入框里输入 jQuery 语句，单击“应用”按钮就可以查看效果。可以在输入框中输入多条语句。单击“应用”按钮后语句会按照顺序分别执行。



图 9-2 jQuery 动画实验室

jQuery 动画实验室中提供了两个可供操作的 DOM 元素：

```
<div id="holder">
  <br />
  <div id="myDiv" style="border:solid 1px #000000;">会动的图层</div>
</div>
```

如果多次单击“应用”按钮，会发现这些动画会有顺序地分别执行。这就是 jQuery 的动画队列，也将在后面讲解到。

### 9.1.4 jQuery 动画时间参数

大部分动画函数，都会有一个作用时间参数。比如：


```
.show( [duration], [ callback ] )
```



“duration”参数就是动画的作用时间。可以是数字或者字符串。如果 duration 是数字，则表示动画持续时间的毫秒数。数值越小，表示动画完成的时间越快；如果 duration 是字符串，那么可以使用以下两个有效值：fast 和 slow，等同于数字的 200 和 600。每一个字符串对应的数字，保存在“jQuery.fx.speeds”对象中。

如图 9-3 所示，除了 fast 和 slow，还有一个“\_default”值。实验证明，此值也是可以使用的：

```
$("#myImg").show().hide("_default");
```



Property	Value
_default	400
fast	200
slow	600

图 9-3 duration()函数的 string 类型值

“\_default”实际上是 jQuery 内部设置的默认值，如果传递了一个 jQuery 不支持的字符串，比如“Orz”，则会使用“\_default”设置的动画时间。比如：

```
$("#myImg").fadeOut(); //等同于$("#myImg").fadeOut(400);
```

当然，也可以为 jQuery.fx.speeds 对象添加新的属性，然后使用：

```
jQuery.fx.speeds["slower"] = 2000;  
$("#myImg").toggle("slower");
```

## 9.1.5 jQuery 动画回调函数

大部分动画函数，都会有一个回调函数的参数。比如：

```
.show( [duration], [callback] )
```

“callback”是动画的回调函数。当动画完成时，会调用 callback()函数，调用时不会传入参数。callback()函数会为每一个完成动画的元素调用一次，而不是一共只调用一次。

下面是 jQuery 动画实验室元素的 HTML 片段：

```
<div id="holder">  
  <br />  
  <div id="myDiv" style="border:solid 1px #000000;">会动的图层</div>  
</div>
```

调用如下语句：

```
$("#holder *").toggle(200,function(){ alert("finish");});
```

此时，会显示三次“finish”，因为选择器“#holder \*”会选中 holder 中所有的元素，包括一个 img 元素、一个 br 元素和一个 div 元素。

## 9.2 基础动画函数

使用基础动画函数 show、hide 可以快速地显示和隐藏元素，toggle 可以切换元素的显示状态。下面是基础动画函数的所有函数签名，如表 9-2 所示。





表 9-2 基础动画函数签名列表

函数签名	返回值	加入版本	参数说明
<code>.show( [duration], [ callback ] )</code>	jQuery	1.0	duration: 字符串或数字, 表示动画持续的时间 callback: 动画完成时执行的函数
<code>.hide( [duration], [ callback ] )</code>	jQuery	1.0	duration: 字符串或数字, 表示动画持续的时间 callback: 动画完成时执行的函数
<code>.toggle( [ duration ], [ callback ] )</code>	jQuery	1.0	duration: 字符串或数字, 表示动画持续的时间 callback: 动画完成时执行的函数
<code>.toggle( showOrHide )</code>	jQuery	1.3	showOrHide: 一个 boolean 值, true 表示显示, false 表示隐藏

## 9.2.1 基础动画实例

现在可以在 jQuery 动画实验室页面, 演示各个函数的用法。  
直接隐藏对象 (无动画效果)。

```
$("#myImg").hide();
```

直接显示对象 (无动画效果)。

```
$("#myImg").hide().show();
```

直接切换显示状态 (无动画效果)。

```
$("#myImg").toggle();
```

首先显示使用渐变动画隐藏元素, 接着使用渐变动画显示元素。渐变的时间都是 1 秒 (1000 毫秒)。

```
$("#myDiv").hide(1000).show(1000);
```

如果想切换对象的显示状态, 比如上面的代码是在显示和隐藏之间切换, 则可以使用 toggle() 函数实现相同的效果:

```
$("#myDiv").toggle(1000);
```

### 注意

img 元素的 myImg 使用 show 时, 则渐变效果存在问题。可以将图片元素放在 div 中, 对容器 div 使用渐变动画就不存在问题了。

在渐变动画结束时, 显示提示语句:

```
$("#holder").show().hide('slow',function(){alert("hide complete!");});
```

## 9.2.2 基础动画详解

show() 和 hide() 函数虽然称为“基础动画函数”, 实际上是综合了滑动动画和渐变动画两种渐变效果的综合函数, 如果传递了 duration 参数, 则会用综合了滑动和渐变两种效果的动画显示或隐藏对象。

show() 函数和 hide() 函数是有关系的。当使用 hide() 函数时, 会首先记录元素的显示状态。比如简单修改一下 jQuery 动画实验室的元素:



```

```

设置图片是 display 样式为 inherit，调用语句如下：

```
$("#myImg").hide(1000).show(1000);
```

图片经历了先隐藏再显示的过程，最后 display 的属性还是 inherit。这是因为在使用 hide() 函数时，会将此元素的 display 属性保存起来。

```
jQuery.data(this[i], "olddisplay", jQuery.css(this[i], "display"));
```

当再次使用 show() 函数显示时，会首先读取原 display 属性。

```
var old = jQuery.data(this[i], "olddisplay");
```

目前在 1.4 版本的 jQuery 中，存在着一个 bug。假设有一个图片元素：

```

```

对其应用下列语句：

```
alert($("#myImg")[0].style.display); // output: "inherit"
$("#myImg").show();
alert($("#myImg")[0].style.display); // output: ""
$("#myImg").hide().show();
alert($("#myImg")[0].style.display); // output "inline"
```

最后 img 元素的 display 属性变成了“inline”！原因是此 img 元素没有使用 hide() 函数保存它的历史属性。而使用 show() 函数显示时，存在着将 display 属性设置为空的 bug。如果一个元素的 display 是空，使用 jQuery 的 hide() 函数获取 display 的值是“inline”！关于此 bug 的原因，在于下列语句：

```
elem.ownerDocument.defaultView.getComputedStyle( elem, null );
if ( computedStyle ) {
    ret = computedStyle.getPropertyValue( name );
}
```

jQuery 内部，使用了 getComputedStyle() 函数用来获取样式的计算值。而当一个元素的 display 属性是空时，其计算值是“inline”！

所以应尽量避免在对元素 hide 前使用 show() 函数。“先 hide 再 show”就不会出现上述问题。

## 9.3 渐变动画函数

渐变动画函数，可以实现对象透明化渐变效果。fadeIn() 是使对象渐变显示，fadeOut() 是使对象渐变隐藏，fadeTo() 可以指定渐变后的目标透明度。下面是渐变动画函数重载列表，如表 9-3 所示。

表 9-3 渐变动画函数签名列表

函数签名	返回值	加入版本	参数说明
.fadeIn( [ duration ], [ callback ] )	jQuery	1.0	duration: 字符串或数字，表示动画持续的时间 callback: 动画完成时执行的函数



(续表)

函数签名	返回值	加入版本	参数说明
<code>.fadeOut( [ duration ], [ callback ] )</code>	jQuery	1.0	duration: 字符串或数字, 表示动画持续的时间 callback: 动画完成时执行的函数
<code>.fadeTo( duration, opacity, [ callback ] )</code>	jQuery	1.0	duration: 字符串或数字, 表示动画持续的时间 callback: 动画完成时执行的函数 opacity: 0 到 1 的一个值, 对象动画的目标透明度

### 9.3.1 渐变动画实例

首先, 依然通过实例快速上手渐变动画函数。在动画实验室可以快速地应用这些实例。在元素缓慢渐变消失后, 快速渐变显示:

```
$("#myImg").fadeOut("slow").fadeIn("fast");
```

上面的代码等同于:

```
$("#myImg").fadeOut(600).fadeIn(200);
```

使用 400 毫秒的动画时间隐藏和显示元素:

```
$("#myImg").fadeOut().fadeIn();
```

元素消失后, 显示提示语句:

```
$("#myImg").fadeOut(function(){alert("finished!");});
```

使元素在 3 秒内变得半透明:

```
$("#myImg").fadeTo(3000,0.5);
```

立刻取消元素的透明度 (设置为不透明):

```
$("#myImg").fadeTo(0,1);
```

### 9.3.2 渐变动画详解

渐变动画函数的使用与基本动画函数基本类似。渐变动画只能控制元素的透明度。`fadeIn` 和 `fadeOut` 分别控制元素渐变显示和渐变隐藏。对于一个使用 `fadeIn()` 函数显示的元素, 可以使用 `hide()` 函数将其隐藏:

```
$("#myImg").hide().fadeIn(1000);
```

`fadeIn()`、`fadeOut()` 与 `fadeTo()` 是不同的。`fadeIn()` 和 `fadeOut()` 会改变元素的显示状态, 也就是不仅仅改变元素的 `opacity` 属性, 同时还改变元素的 `display` 属性。

使用 `fadeIn()` 函数前的元素状态:

```

```

使用 `fadeIn()` 函数后的元素状态:

```

```

使用 `fadeOut()` 函数前的元素状态:

```

```





使用 fadeOut()函数后的元素状态:

```

```

可以看出, fadeIn()函数改变元素 display 样式的效果和 show()函数相同, fadeOut()函数改变元素 display 样式的效果和 hide()函数相同。使用 fadeOut()隐藏对象时,会将对象的 display 属性值保存,当再次使用 show()或者 fadeIn()函数时,会找回元素的显示状态:

```
$("#myImg").fadeOut().fadeIn(); //元素的 display 属性不变
```

fadeTo()函数则完全不同, fadeTo 函数不改变元素的 display 属性,只改变元素的透明度属性 “opacity”:

```
$("#myImg").fadeTo("slow",0.5);
```

使用上面的语句后,元素会变成半透明:

此时,使用 fadeIn 函数,元素并不会有任何变化。使用 fadeOut()函数还可以让元素消失,但是再次使用 fadeIn()或者 show()函数让对象显示时,元素依然是半透明的,如图 9-4 所示。

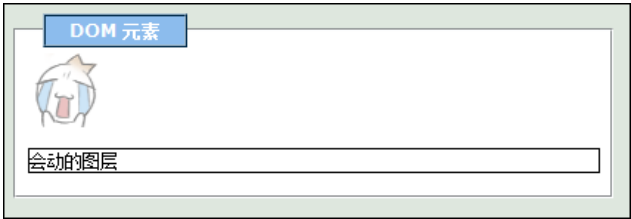


图 9-4 使用 fadeTo()函数实现半透明效果

用 fadeTo()函数改变元素的透明度时,如果 duration 传递了 0,则没有动画:

```
$("#myImg").fadeTo(0,0.1);
```

上面的语句,等同于:

```
$("#myImg").css("opacity", 0.1);
```

## 9.4 滑动动画函数

滑动动画函数,用于改变元素的高度,实现“滑动效果”。下面是滑动动画函数所有的函数重载,如表 9-4 所示。

表 9-4 滑动动画函数签名列表

函数签名	返回值	加入版本	参数说明
.slideDown ( [ duration ], [ callback ] )	jQuery	1.0	duration: 字符串或数字,表示动画持续的时间 callback: 动画完成时执行的函数
.slideUp ( [ duration ], [ callback ] )	jQuery	1.0	duration: 字符串或数字,表示动画持续的时间 callback: 动画完成时执行的函数
.slideToggle( [ duration ], [ callback ] )	jQuery	1.0	duration: 字符串或数字,表示动画持续的时间 callback: 动画完成时执行的函数

### 9.4.1 滑动动画实例

依然是通过实例，快速上手滑动动画函数，具体请参看下面的实例：  
让元素缓慢滑动消失后，快速滑动显示：

```
$("#holder").slideUp("slow").slideDown("fast");
```

上面的代码等同于：

```
$("#holder").slideUp(600).slideDown(200);
```

使用 400 毫秒的动画时间隐藏和显示元素：

```
$("#holder").slideUp().slideDown();
```

元素消失后，显示提示语句：

```
$("#myImg").slideUp()(function(){alert("finished!");});
```

切换元素的显示状态，并在动画完成后实现提示语句：

```
$("#holder").slideToggle(function(){alert("finished!");});
```

### 9.4.2 滑动动画详解

如果元素已经是隐藏状态，执行：

```
$("#myImg").slideUp()(function(){alert("finished!");});
```

会发现没有任何动画效果，直接显示提示框“finished!”，即如果元素的状态不需要执行动画效果，则会立刻执行 `callback()` 回调函数。

函数的其他特性，与基础动画函数和渐变动画函数基本相同，所以不再重复类似讲解。

## 9.5 自定义动画函数

自定义动画函数包括了所有动画效果的高级应用和高级技巧，如表 9-5 所示。

表 9-5 自定义动画函数分类（Effects/Custom）函数列表

函数签名	返回值	加入版本	函数说明
<code>.animate()</code>	jQuery	1.0	执行自定义的动画效果
<code>.stop()</code>	jQuery	1.2	停止匹配元素当前正在执行的动画效果
<code>jQuery.fx.off</code>	Boolean	1.3	取消全部动画效果
<code>.queue()</code>	Array 或 jQuery	1.2	根据函数签名不同，可以获取指定队列的函数集合（返回 Array），或者操作某一个队列（返回 jQuery）
<code>.dequeue()</code>	jQuery	1.2	执行匹配元素指定队列中的下一个函数
<code>.clearQueue()</code>	jQuery	1.4	清除队列中的所有未执行函数
<code>.delay()</code>	jQuery	1.4	延时执行队列中的函数

表 9-5 列出了在自定义动画分类，也就是 jQuery 官方文档中 Effects 总分类下 Custom 子分类中的所有函数。自定义函数是 jQuery 动画的高级内容。下面将由浅入深，从基础的队列开始，来学习 jQuery 的自定义动画函数。

### 9.5.1 jQuery 队列

对某一个元素调用动画函数时，会发现动画是按照顺序执行的：

```
$("#holder").hide(1000).fadeIn();
```

也许在一个 jQuery 链上顺序执行不奇怪，但是可以同时运行两次语句：

```
$("#holder").hide(1000).fadeIn();
$("#holder").hide(1000).fadeIn();
```

这时将会发现这四个动画函数都是排队执行的！

如果没有队列，hide()函数和 fadeIn()函数并不会彼此等待，而是立刻全部执行完毕，但是 jQuery 中的动画效果会排队作用在元素上，“排队”效果就是通过 jQuery 中的队列实现的。

在 jQuery 内部，所有的动画函数都会排列在名为“fx”的队列中。jQuery 中提供了下列和队列有关的函数，如表 9-6 所示。

表 9-6 队列所有相关函数签名列表

函数签名	返回值	加入版本	参数说明	函数说明
.queue( [ queueName ] )	Array	1.2	queueName: 队列的名字，默认是动画队列的名字“fx”	返回指定名字队列的函数数组
.queue( [ queueName ], newQueue )	jQuery	1.2	queueName: 队列的名字，默认是动画队列的名字“fx” newQueue: 一个函数数组，用于替换 queueName 队列的内容	替换指定的队列内容
.queue( [ queueName ], callback( next ) )	jQuery	1.2	queueName: 队列的名字，默认是动画队列的名字“fx” callback( next ): 要添加到队列中的函数	为指定队列添加一个函数
.dequeue( [ queueName ] )	jQuery	1.2	queueName: 队列的名字，默认是动画队列的名字“fx”	执行指定队列中的某一个函数
.delay( duration, [ queueName ] )	jQuery	1.4	duration: 字符串或数字，表示延时的时间 queueName: 队列的名字，默认是动画队列的名字“fx”	让某一个队列的执行延迟

jQuery 中的 queue 和 dequeue 是一组很有用的方法，它们对于一系列需要按次序运行的函数特别有用。特别是 animate 自定义动画、AJAX，以及 timeout 等需要一定时间执行的函数。

在 jQuery 1.4 版本中，还加入了 delay()函数可以使队列的执行延时。

下面通过一个实例，了解 queue()函数：

【代码路径：jQueryStorm.Web/chapter9/ Demo-Queue.htm】

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<script src="../../Static/common/js/jquery-1.4.2.min.js"></script>
</head>
```



```

<body>
  <button id="show">
    Go</button>
  <div id="holder" style="position: relative; height: 60px;">
    <div id="myDiv" style="border: solid 1px #000000; width: 50px; height: 50px; position: absolute;">
      会动的图层
    </div>
  </div>
  <div id="divMsg" style="border: solid 1px; width:100px;">
    [console]
  </div>
  <script>
    function go() {
      $("#divMsg").html($("#myDiv").queue().length);
    };
    $("#show").click(function(e) {
      //执行一系列动画函数
      $("#myDiv").show("slow", go);
      $("#myDiv").animate({ left: '+=200' }, 2000, go);
      $("#myDiv").slideToggle(1000, go);
      $("#myDiv").slideToggle("fast", go);
      $("#myDiv").animate({ left: '-=200' }, 1500, go);
      $("#myDiv").hide("slow", go);
      go();
    });
  </script>
</body>
</html>

```

实例效果如图 9-5 所示。

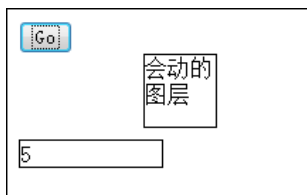


图 9-5 利用 queue()函数获取队列个数实例

单击“Go”按钮后，“会动的图层”将开始移动，并且在下面的 div 中显示目前队列中的函数数目。从“6”开始，最后变为“0”。这是因为一开始向队列中添加了 6 个动画函数，并且逐一执行。

队列实际上是一个函数数组，要获取队列中函数数组，上例中使用的是.queue( [ queueName ] )函数，其中 queueName 参数如果省略，则表示获取默认的动画队列 fx。

可以通过.queue( [ queueName ], newQueue )设置一个队列：

```

var myQueue = [
  function() { alert("1"); go(); $(this).dequeue(); },
  function() { alert("2"); go(); $(this).dequeue(); },
  function() { alert("3"); go(); $(this).dequeue(); },
];
$("#myDiv").queue(myQueue);

```

其中 newQueue 参数是一个函数数组。

除了传递函数数组替换全部队列，还可以为队列添加一个函数。比如：

```

$("#myImg").slideUp();
$("#myImg").queue(function(){alert('add a function');$(this).dequeue();});

```



上面的代码等同于：

```
$("#myImg").slideUp(function(){alert('add a function');$(this).dequeue();});
```

特别需要注意，添加的函数一定要执行 `dequeue()` 函数。`dequeue()` 函数相当于将当前队列中的最前面的一个函数取出，并执行。

下面的例子中，会首先执行 `slideUp()` 函数滑动隐藏，然后显示提示语句，再执行 `slideDown()` 函数。

```
$("#myImg").slideUp();  
$("#myImg").queue(function(){alert('add a function');$(this).dequeue();});  
$("#myImg").slideDown();
```

如果将 `dequeue()` 函数去掉：

```
$("#myImg").slideUp();  
$("#myImg").queue(function(){alert('add a function');});  
$("#myImg").slideDown();
```

在显示提示文字“add a function”后，`slideDown()` 函数不再执行。

在 jQuery 1.4 版本中，当调用队列函数时，多了一个参数：

```
.queue( [ queueName ], callback( next ) )
```

`next` 参数是一个函数，调用 `next` 等同于手工调用 `dequeue()` 函数：

```
$("#myImg").slideUp();  
$("#myImg").queue(function(next){alert('add a function'); next(); });  
$("#myImg").slideDown();
```

上面的例子中，`slideDown()` 函数会执行。

在使用队列时，也常会使用 `delay()` 函数。`delay()` 可以为某一个队列添加延时。比如：

```
$("#myImg").slideUp().delay(1000).slideDown();
```

上面的例子中，当 `slideUp` 执行完毕后，会等待 1 秒钟，再执行 `slideDown()` 函数。

`delay()` 函数的签名是：

```
.delay( duration, [ queueName ] )
```

其中 `duration()` 函数和动画函数相同，可以使用字符串或者数字。`queueName()` 表示队列名称。

`delay()` 函数是 jQuery 1.4 版本中加入的，功能虽然类似于原始 JavaScript 中的 `setTimeout()` 函数，但是通常只用于队列中，并不能完全替代 `setTimeout` 的所有场景。

### 9.5.2 动画全局开关

有时页面上有许多的元素，每个元素都有自己的动画队列，jQuery 提供了一个全局的属性：

```
jQuery.fx.off
```

来关闭所有的动画效果。

当此属性被设置为 `true` 时，所有的动画函数都会直接将元素设置为最后的状态，不会有动画效果。



通常在下列情况下，会考虑使用 jQuery.fx.off 属性：

- ❑ 动画效果出现问题。
- ❑ 用户设备显示动画有性能问题，通常由于设备较老。

下面通过实例演示此属性的使用。

【代码路径：jQueryStorm.Web/chapter9/ Demo-Off.htm】

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<style>
div { width:50px; height:30px; margin:5px; float:left;
background:green; }
</style>
<script src="../../Static/common/js/jquery-1.4.2.min.js"></script>
</head>
<body>
<p><input type="button" value="切换图层显示状态"/> <button>更改 jQuery.fx.off 属性</button></p>
<div></div>
<script>

$("button").click(function(e) { $.fx.off = !$.fx.off; });
$("input").click(function() {
    $("div").toggle("slow");
});
</script>
</body>
</html>
```

运行效果如图 9-6 所示。

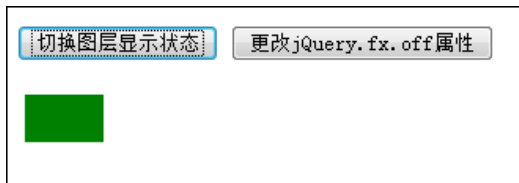


图 9-6 jQuery.fx.off 属性实例

在页面加载后，单击“切换图层显示状态”按钮，绿色的图层会用动画效果显示和隐藏。单击“更改 jQuery.fx.off 属性”按钮，会切换 jQuery.fx.off 的属性值。当单击一次即 jQuery.fx.off 设置为 true 时，绿色图层的显示和隐藏都没有了动画效果。

### 9.5.3 停止元素动画

除了全局的动画开关，还可以使用 stop() 函数停止元素的动画效果。下面是 stop() 函数的签名，如表 9-7 所示。

表 9-7 stop() 函数签名

函数签名	返回值	加入版本	参数说明	函数说明
<code>.stop( [ clearQueue ], [ jumpToEnd ] )</code>	jQuery	1.2	<code>clearQueue</code> : 布尔值，默认是 false。表示是否清除队列中的动画函数。 <code>jumpToEnd</code> : 布尔值，默认是 false。表示是否立刻完成动画	停止匹配元素当前的动画效果



stop()函数是作用在元素上的。当使用无参数的 stop()函数时，如果一个动画还在执行，则元素会停留在当前的状态。

比如在 jQuery 实验室上，应用下列语句：

```
$("#myDiv").slideUp(3000);
```

单击“停止”按钮，会对 myDiv 对象调用 stop 语句，可以立刻停止在当前的动画状态上，如图 9-7 所示。

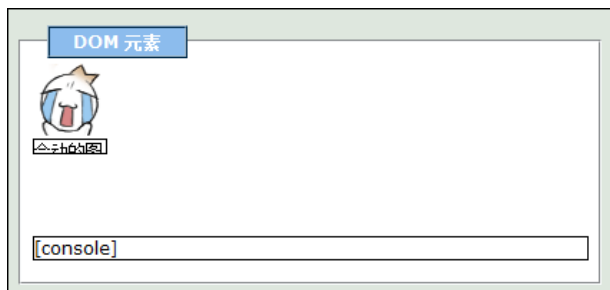


图 9-7 调用 stop()函数

当 stop()函数调用后，动画会停留在当时的状态上，并且动画的 callback()回调函数也不会被调用。

但是默认不带参数的 stop()函数并不总是停止动画的运行。这是因为 stop()函数会停止动画队列当前的函数，如果动画队列中还有函数的话，后面的函数会立刻执行。

比如，下面的语句是使 myDiv 运动一次来回的例子：

```
$("#myDiv").show("slow");
$("#myDiv").animate({ left: '+=200' }, 2000);
$("#myDiv").animate({ left: '-=200' }, 1500);
$("#myDiv").hide("slow");
```

在正常情况下，图层会运动到指定的位置，然后再返回。

但是如果在运动过程中，比如在

```
$("#myDiv").animate({ left: '+=200' }, 2000);
```

过程中，运行：

```
$("#myDiv").stop();
```

会发现图层立刻向回移动，也就是开始执行：

```
$("#myDiv").animate({ left: '-=200' }, 1500);
```

所以，如果希望 stop()函数不仅仅停止当前动画，同时清除队列时，可以传递 clearQueue 参数：

```
$("#myDiv").stop(true, false);
```

## 注意

clearQueue 和 jumpToEnd 参数在使用时最好同时传递。因为这两个的类型相同，如果只传递一个则无法判断函数重载。



当 `clearQueue` 属性设置为 `true` 时，动画队列中的所有函数都会被清除。

当 `jumpToEnd` 属性设置为 `true` 是，元素不是停止在当前状态，而是立刻变成动画执行后的状态。类似于将 `jQuery.fx.off` 设置为 `true` 时动画直接跳到最后的状态。但是对于已经开始动的动画，`jQuery.fx.off` 属性是没有效果的，而使用 “`stop(false, true)`” 语句是有效果的。

### 9.5.4 自定义动画效果

`animate()` 函数签名如表 9-8 所示。

表 9-8 `animate()` 函数签名

函数签名	返回值	加入版本	参数说明	函数说明
<code>.animate( properties, [ duration ], [ easing ], [ callback ] )</code>	jQuery	1.0	<code>properties</code> : CSS 属性集合，动画的目标效果。 <code>duration</code> : 字符串或数字，表示延时的时间 <code>easing</code> : 要使用的擦除效果的名称（需要插件支持）默认 jQuery 提供 “linear” 和 “swing” <code>callback</code> : 在动画完成时执行的函数	用于创建自定义动画的函数
<code>.animate( properties, options )</code>	jQuery	1.0	<code>properties</code> : CSS 属性集合，动画的目标效果。 <code>options</code> : 一组包含动画选项的值的集合	用于创建自定义动画的函数

觉得滑动、淡入淡出和基础动画函数都太简单？不要紧，jQuery 提供了 `animate()` 函数可以用来实现复杂的动画效果。

`animate()` 函数必须传递的参数只有一个：CSS 属性集合。这和使用 `.css()` 函数时传递的集合是一样的，但是要求更严格。支持的 CSS 属性，必须要使用数字值。比如 `width`、`height`、`left` 等。还记得前面使用 `animate()` 函数在 jQuery 动画实验室，移动 `myDiv` 图层的例子吗？下面通过稍微的修改，使 `myDiv` 在水平移动的同时，还改变大小：

```
$("#myDiv").animate({left:"+=200",height:"+=200",width:"+=200"},2000);
```

效果如图 9-8 所示。

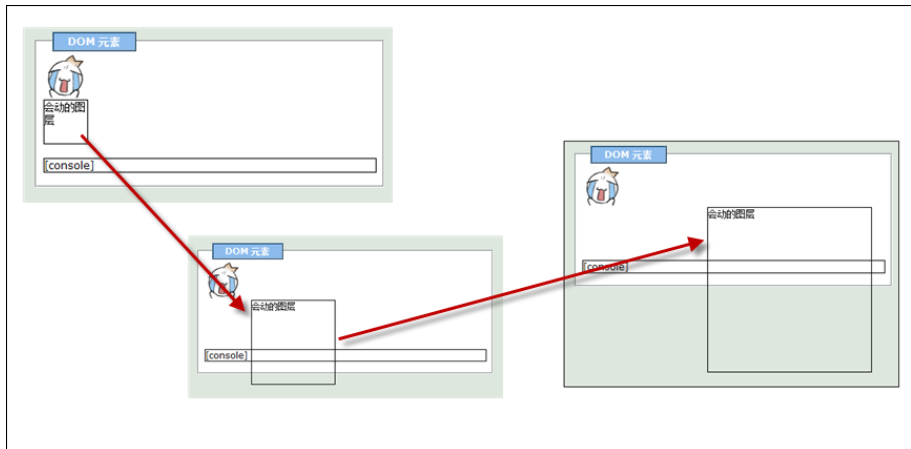


图 9-8 `animate()` 简单动画

所有指定的属性必须用骆驼形式，比如用 `marginLeft` 代替 `margin-left`。

属性值除了数字，还支持 “em” 和 “%” 单位。如果使用的是 “hide”、“show” 或 “toggle”



这样的字符串值，则会为该属性调用默认的动画形式：

```
$("#myDiv").animate({left:"+=200",height:"toggle"},2000);
$("#myDiv").animate({left:"-=200",height:"toggle"},2000);
```

另外属性值还可以是相对值。可以在数字前面添加“+=”或“-=”。“+=”表示在当前值基础上增加，“-=”表示在当前值基础上减少。

上面例子中，将位置、宽度和高度都增加了 200 像素，并且在 2 秒内完成增加这些像素的动态效果。

**duration** 和 **callback** 参数的用法和其他动画函数相同，分别设置动画时间和动画完成时的回调函数。再次提醒，**callback()**函数是没有任何参数的，其中的 **this** 指向当前的 DOM 元素，**callback()**函数会作用在每一个元素上而不是集合上。

**easing** 参数，是指实现属性渐变动画的算法函数。比如透明度 **opacity** 从 0 变为 1，就需要一种算法实现动画效果。在 jQuery 中提供了实现“swing”和“linear”的两种 easing 算法。有很多的插件可以扩展 easing。

在 jQuery1.4 版本中，每一个 CSS 属性都可以单独地指定所要使用的 easing 算法。

```
$('#myDiv').animate({
    width: ['toggle', 'swing'],
    height: ['toggle', 'swing'],
    opacity: 'toggle'
}, 5000, 'linear', function() {
    $(this).after('<div>Animation complete.</div>');
});
```

如果某一个属性没有指定 easing 算法，则使用“swing”。上面的例子，指定了 easing 参数是“linear”，所以 **opacity** 的渐变效果将使用“linear”，**width**、**height** 的渐变效果使用“swing”。如果没有传递 easing 参数，则 **opacity** 会使用默认值 swing。

在 **animate()**的第二个重载函数中：

```
.animate( properties, options )
```

可以传递一个 **options** 参数。**options** 是一个设置项属性集合，包含下列属性，如表 9-9 所示。

表 9-9 animate()函数 options 参数属性列表

属性名称	值类型	说 明
duration	string/numeric	字符串或数字，表示延时的时间
easing	string	要使用的 easing 组件名称
complete	function	当动画执行完毕后的回调函数
step	function	在动画每一步都执行的回调函数
queue	Boolean	默认为 true，设置为 false 此动画将不进入动画队列立刻执行
specialEasing	map object	一个包含名/值对的对象，属性名是 CSS 属性名称，值是 easing 组件名称

其中 **specialEasing** 是 jQuery1.4 版本才加入的属性，通过此属性也可以为每一个 CSS 属性指定 easing 算法。

```
$('#myDiv').animate({
    width: 'toggle',
    height: 'toggle'
```



```

}, {
  duration: 5000,
  specialEasing: {
    width: "linear",
    height: "easeOutQuad"
  },
  complete: function() {
    $(this).after('<div>Animation complete.</div>');
  }
});

```

上面的例子中，将使用“linear”算法变换宽度，使用“easeOutQuad”算法变换高度。jQuery 默认不含有“easeOutQuad”这个 easing 算法，需要额外下载。在本书程序目录中的：

【代码路径：jQueryStorm.Web /Static/Common/js/jquery.easing.plugin.js】

可以找到此 easing 算法插件。里面包括很多的 easing 算法。

animate()是最复杂的动画函数，下面通过实例，熟悉 animate 的各种特性和用法。

使用“%”、“in”和“em”单位：

```

$("#myDiv").animate({
  width: "70%",
  opacity: 0.4,
  marginLeft: "0.6in",
  fontSize: "3em",
  borderWidth: "10px"
}, 1500);

```

效果如图 9-9 所示。



图 9-9 使用不同的数值单位

使用 animate()函数替换 fadeIn()函数。

```

$("#myDiv").hide().css("opacity",0).show().animate({
  opacity: 1
}, 1000);

```

使用 animate()函数替换 slideUp()函数。

```

$("#myDiv").animate({
  height: "hide"
}, 1000);

```

使用 animate()函数替代 slideToggle()函数。

```

$("#myDiv").animate({
  height: "toggle"
}, "fast");

```



注意将 height 属性设置为 0 和 “toggle” 是有区别的。设置为 “toggle” 会在渐变最后将元素隐藏，而渐变为 0 则不会隐藏元素。

实现类似 toggle() 函数的效果。

```
$("#myDiv").animate({
    height: "toggle",
    width: "toggle",
    opacity: "toggle"
}, "slow");
```

将一个动画排除在队列之外。这样两个动画效果会同时执行。

```
$("#myDiv")
    .animate({
        height: "40%"
    }, 2000)
    .animate({
        left: "50px", opacity: 1
    }, { duration: 2000, queue: false});
```

下面实现一个自定义坠落动画的效果。

```
$("#myDiv").animate(
{
    "opacity": "hide",
    "top": $(window).height() - $("#myDiv").height() - ($("#myDiv").position().top
    }, 600);
```

实现效果如图 9-10 所示。

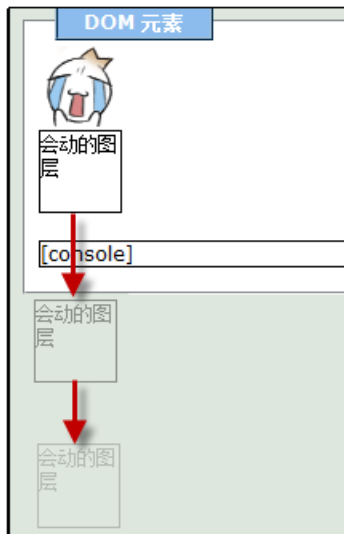


图 9-10 坠落动画效果

实现物体消散效果。

```
$("#myDiv").animate(
{
    "opacity": "hide",
    "width": $(window).width()-100 ,
    "height": $(window).height()-100
}, 500);
```



## 9.6 小结

本章学习了使用 jQuery 的动画函数，使用 jQuery 可以让页面生动起来。曾经那些看似高深的页面特效，现在使用 jQuery 都可以轻松实现。

本章的 9.1 讲解了 jQuery 动画的基础知识，在各个分类中常会用到，所以作为基础应该牢记 duration、回调函数的使用。

可以使用最简单的基础动画函数创建最常用的综合显示效果，也可以使用最基础的滑动、淡入淡出函数创建基本的动画样式。

而在学习了自定义动画后，发现使用最复杂的 animate() 函数可以创造更多的动画效果。但是也要注意队列在动画中起的作用。用好队列，在事务处理和 AJAX 函数中都会有用武之地。

本章介绍的都是 jQuery 动画的相关函数，侧重于掌握函数的使用。动画效果最后往往应用于页面的某些“控件”上。其中 jQuery UI 就是在应用了 jQuery 动画，在 jQuery 类库的基础之上开发出来的 UI 界面库，可以提供更加绚丽的页面特效、简便易用的页面控件等。第 12 章将介绍如何使用 jQuery UI 让页面更加丰富多彩。

为何没有安排在第 10 章讲解 jQuery UI？因为 jQuery 的类库还没有完全讲解完。地基打得越牢固，大厦才能建得越高。所以第 10 章将首先介绍 jQuery 工具函数。





## 第 10 章



# jQuery 工具函数

工具函数的概念对于开发人员来说并不陌生。如果一个函数是和业务逻辑无关、相对独立、可以被简单调用、实现了某一个功能，则这类函数被称为工具函数。工具函数通常是静态方法，使用时不需要实例化对象。工具函数在系统中扮演着底层劳动者的角色。

jQuery 中有许多有用的工具函数，并且将工具函数分类称为“Utilities”。本章将全面细致地介绍 jQuery 中的工具函数。

但是 jQuery 的工具函数相对于其他的脚本类库，显得十分单薄。这是由于 jQuery 的“轻量级”导致的。在 jQuery 已经存在的工具函数之上，开发人员可以根据自己的需要不停地补充和完善自己的工具函数库。认真学习 jQuery 的工具函数，无论对日常的业务逻辑开发，或是插件开发、脚本控件开发，都十分有帮助。



## 10.1 jQuery 工具函数基础

在深入了解 jQuery 提供的工具函数之前，下面先概括地了解一下 jQuery 工具函数。

### 10.1.1 工具函数说明

在最新的 jQuery 官方在线文档 [api.jquery.com](http://api.jquery.com) 中，Utilities 分类下存放着所有的 jQuery 工具函数，并且没有再区分子类。

Utilities 分类中，有所有和队列相关的函数，第 9 章已经讲解了 jQuery 的队列，所以在本章中不包含与 jQuery 队列相关的内容。

jQuery 工具函数个数很多并且没有再分类，在接下来讲解的时候将按照功能将工具函数分类讲解，这样有助于理解。

除了队列函数，其他所有的 jQuery 工具函数都是静态函数。

### 10.1.2 jQuery 工具函数概览

在详细地学习 jQuery 工具函数之前，先通过表 10-1 来看看 jQuery 都有哪些工具函数。

表 10-1 Utilities 分类函数列表

函数名称	函数说明
jQuery.boxModel	在 jQuery 1.3 中不建议使用。当前页面中浏览器是否使用标准盒模型渲染页面。建议使用 jQuery.support.boxModel 代替。W3C CSS 盒模型
jQuery.browser	在 jQuery 1.3 中不建议使用。浏览器内核标识。依据 navigator.userAgent 判断
jQuery.contains()	判断一个 DOM 结点是否在另外一个 DOM 结点中
jQuery.data()	保存数据
jQuery.each()	遍历对象
jQuery.extend()	将多个对象属性合并成一个
jQuery.globalEval()	全局执行 JavaScript 代码
jQuery.grep()	使用过滤函数过滤数组元素
jQuery.inArray()	搜索某一个值在集合中的索引位置（如果没有找到则返回-1）
jQuery.isArray()	测试对象是否为数组
jQuery.isEmptyObject()	检查对象是否是空（无属性）
jQuery.isFunction()	判断对象是否是函数
jQuery.isPlainObject()	检查是否是链式对象（使用“{}”或者“new Object”创建的对象）
jQuery.isXMLDoc()	判断一个 DOM 结点是否包含在 XML 文档对象中（或者本身就是 XML 文档对象）
jQuery.makeArray()	将类数组对象转换为 Javascript 原生的数组对象
jQuery.map()	转换数组或类数组对象中的所有元素
jQuery.merge()	将第二个数组的元素添加到第一个数组中
jQuery.noop()	一个空函数
jQuery.parseJSON	将 JSON 串转化为 JavaScript 对象
jQuery.proxy()	jQuery 1.4 新增。返回一个新函数，并且这个函数始终保持了特定的作用域
jQuery.removeData()	移除使用 Data 绑定的数据
jQuery.support	jQuery 1.3 新增。一组用于展示不同浏览器各自特性和 bug 的属性集合



(续表)

函数名称	函数说明
jQuery.trim()	去掉字符串起始和结尾的空格
jQuery.unique()	删除数组中重复元素。只处理删除 DOM 元素数组，而不能处理字符串或者数字数组

## 10.2 浏览器特性检测

脚本开发最头痛的问题就是跨浏览器。同样的脚本在不同的浏览器中常常得到不同的效果。通常的做法是针对不同的浏览器，进行不同的逻辑处理。jQuery 提供了浏览器特性检测函数，用来检测浏览器的特性，方便开发人员实现跨浏览器的脚本。

### 10.2.1 浏览器特性检测的演变

在 jQuery 1.3 版本之前，jQuery 提供了 jQuery.boxModel 和 jQuery.browser 用来检测浏览器的盒式模型和浏览器类型。jQuery 官方明确地表示，不推荐再继续使用这两个属性。推荐使用 jQuery.support 属性来检测浏览器特性，如表 10-2 所示。

表 10-2 浏览器特性相关属性

属性名称	加入版本	说 明
jQuery.support	1.3	推荐使用
jQuery.browser	1.0	不推荐使用
jQuery.boxModel	1.0	已废除

jQuery.boxModel 的功能已经被集成到了 jQuery.support 属性中，jQuery.boxModel 原本用于检测浏览器的盒式模型。后面会讲解什么是盒式模型。jQuery.boxModel 已经被否决，在以后的版本中有可能被删除。

jQuery.browser 用来检测当前的浏览器类型，根据 navigator.userAgent 获取浏览器的内核标识。与 jQuery.boxModel 不同，jQuery.browser 在以后也不会被移除，并且在 jQuery 1.4 中还加入了新的功能。但是使用判断浏览器类型来实现跨浏览器支持的脚本是不被推荐的做法。因为目前浏览器众多，每一种浏览器的某一个版本，都可能拥有特殊的属性。如果针对浏览器和版本，会写出这样的代码：

```
if (jQuery.browser.msie) {
    switch (jQuery.browser.version) {
        case "8.0":
            alert("do something at IE8");
            break;
        case "7.0":
            alert("do something at IE7");
            break;
        case "6.0":
            alert("do something at IE6");
            break;
        default:
            alert("default!");
    }
}
```





```
else if (jQuery.browser.mozilla)
{
    alert("do something at FireFox");
}
else if (jQuery.browser.webkit) {
    alert("do something at webkit");
}
```

上面还仅仅处理了 IE、FireFox 和 Chrome，哦，忘记了 Opera！等等，IE9 已经出现了，代码要再加一个分支吗？

这种处理方式不仅分支众多，而且还需要开发人员记住每一个浏览器的特性。

所以 jQuery 在 1.3 版本之后推荐了另外一种方式，判断浏览器是否支持某一个特性。以盒式模型为例：

```
if (jQuery.support.boxModel) {
    //代码部分
}
```

现在开发人员不需要记住究竟哪个浏览器使用了盒式模型，甚至不用关注未来的某个浏览器是否支持盒式模型，这些都交给 jQuery 自己去判断就好了。

可见基于浏览器特性编程要比基于浏览器类型和版本编程好得多。但是有时还是需要知道目前客户端的浏览器信息的，比如一个静态页面希望统计用户的浏览器信息。（如有条件，使用 Tealeaf 等专业统计软件更好）。

现在要开发兼容多浏览器的脚本，首选使用 jQuery.support 属性，如果必须要检测浏览器，可以使用 jQuery.browser。10.2.2 节将详细地讲解 jQuery.browser 属性。

## 10.2.2 检测浏览器类型和版本

jQuery.browser 属性是一个 map 对象，即包含若干属性的对象。在不同的浏览器中 jQuery.browser 是不同的，比如在 IE 中，拥有 msie 和 version 属性，在 firefox 中拥有 mozilla 和 version 属性。

下面的示例将显示访问此页面的 jQuery.browser 属性内容：

【代码路径：jQueryStorm.Web/chapter10/ Demo-jQuery.browser.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>jQuery - 浏览器特性检测</title>
    <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
</head>
<body>
    <div id="divMsg" style="border:solid 1px #000000;width:90%;">[输出窗口]</div>
    <div>
        <button id="btnTest">jQuery.browser</button>
    </div>
    <!-- 尾部脚本块 -->
    <script type="text/javascript">
        var thisPage = {
            initialize: function () { //加载时执行
                this.initializeDom();
```





```

        this.initializeEvent();
    },
    initializeDom: function () { //初始化 DOM
        this.$btnTest = $("#btnTest");
        this.$divMsg = $("#divMsg");
    },
    initializeEvent: function () { //事件绑定
        this.$btnTest.click(function (e) {
            var result = "";
            //使用 jQuery.browser 获取浏览器信息
            for (var item in jQuery.browser) {
                result += item + ":" + jQuery.browser[item] + "<br/>";
            }
            thisPage.console(result);
        });
    },
    console: function (text) {
        this.$divMsg.append("<div>" + text + "</div>");
    }
}
$(thisPage.initialize());
</script>
</body>
</html>

```

在 IE 中，将输出：

```

msie:true
version:8.0

```

在 Firefox 中，将输出：

```

mozilla:true
version:1.9.2.6

```

上面的示例会遍历 jQuery.browser 的所有属性。可见，在不同的浏览器中，jQuery.browser 对象的属性是不同的。version 属性是无论在何种浏览器中，jQuery.browser 都会具有的属性。像 msie 和 mozilla 这种属性是会变化的，下面是 jQuery.browser 可能拥有的属性列表。

- ☐ webkit。
- ☐ safari。
- ☐ opera。
- ☐ msie。
- ☐ Mozilla。

webkit 属性是 jQuery 1.4 添加的，在 Google Chrome 中，会存在 webkit 并且设置为 true。下面是使用 Chrome 时 jQuery.browser 的所有属性，如图 10-1 所示。

在 Chrome 浏览器下仍然具有 safari 属性，也就是说目前 jQuery.browser 无法区分 Chrome 和 safari 浏览器。另外需要注意，jQuery.browser.msie 这类



图 10-1 Chrome 下 jQuery.browser 的属性



属性返回的都是 boolean 值，jQuery.browser.version 的返回值是字符串而不是数字。

### 10.2.3 浏览器特性检测

jQuery.support 属性是一个对象，保存了检测浏览器特性的各种属性，如表 10-3 所示。

表 10-3 jQuery.support 属性列表

属性名称	属性解释
boxModel	如果这个页面和浏览器是以 W3C CSS 盒式模型来渲染的，则等于 true。通常在 IE 6 和 IE 7 的怪癖模式中这个值是 false。在 document 准备就绪前，这个值是 null
cssFloat	如果用 cssFloat 来访问 CSS 的 float 值，则返回 true。目前在 IE 中会返回 false，用 styleFloat 代替
hrefNormalized	如果浏览器从 getAttribute("href")返回的是原封不动的结果，则返回 true。在 IE 中会返回 false，因为它的 URLs 已经常规化了
htmlSerialize	如果浏览器通过 innerHTML 插入链接元素的时候会序列化这些链接，则返回 true，目前 IE 中返回 false
leadingWhitespace	如果在使用 innerHTML 的时候浏览器会保持前导空白字符，则返回 true，目前在 IE、IE7 和 IE8 中返回 false
noCloneEvent	如果浏览器在克隆元素的时候不会连同事件处理函数一起复制，则返回 true，目前在 IE 中返回 false
objectAll	如果在某个元素对象上执行 getElementsByTagName("*")会返回所有子孙元素，则为 true，目前在 IE 7 中为 false
opacity	如果浏览器能适当解释透明度样式属性，则返回 true，目前在 IE 中返回 false，因为它用 alpha 滤镜代替
scriptEval	使用 appendChild/createTextNode 方法插入脚本代码时，浏览器是否执行脚本，目前在 IE 中返回 false，IE 使用 .text 方法插入脚本代码以执行
style	如果 getAttribute("style")返回元素的行内样式，则为 true。目前 IE 中为 false，因为它用 cssText 代替
tbody	如果浏览器允许 table 元素不包含 tbody 元素，则返回 true。目前在 IE 中会返回 false，它会自动插入缺失的 tbody

#### 1. 盒式模型 boxModel

如图 10-2 所示为 W3C 标准中的盒式模型图。

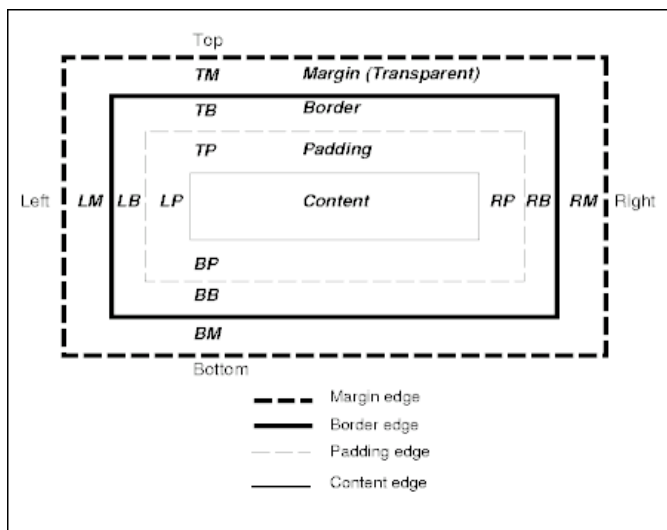


图 10-2 W3C 标准盒式模型

假设如下元素:

```
<style type="text/css">
.boxModel
{
    width:200px;
    height:50px;
    padding:10px;
    border:solid 5px #FF0000;
    background-color:#acacac;
}
</style>
<div id="divBox" class="boxModel">
```

显示效果如图 10-3 所示。

在 CSS 中设定元素宽度为 200px, 下面以此元素为例讲解盒式模式。



图 10-3 盒式模型实例

## 2. W3C 盒式模型

元素的宽度和高度为盒式模型图中的 Context 部分, 不包括 padding、border 和 margin 部分。

目前除了 IE 所有的浏览器都仅支持 W3C 盒式模型。在 W3C 盒式模型中, 示例中包含红框在内的区域内容宽度为  $200+2\times 10+2\times 5=230\text{px}$ , 高度为  $50+2\times 10+2\times 5=80\text{px}$ 。

## 3. IE 盒式模型

设置的宽度包括 padding 和 border, 实际内容宽度  $\text{content Width} = \text{width} - \text{padding} - \text{border}$ 。

在 IE 5.5 及更早的版本中, 使用了此模型。在更高的 IE 版本上如果由于某些原因让浏览器运行在怪异模式下则也会使用此盒式模式, 所以需要在页面上声明正确的 DOCTYPE。

下面是两种盒式模型的对比, 如图 10-4 所示。

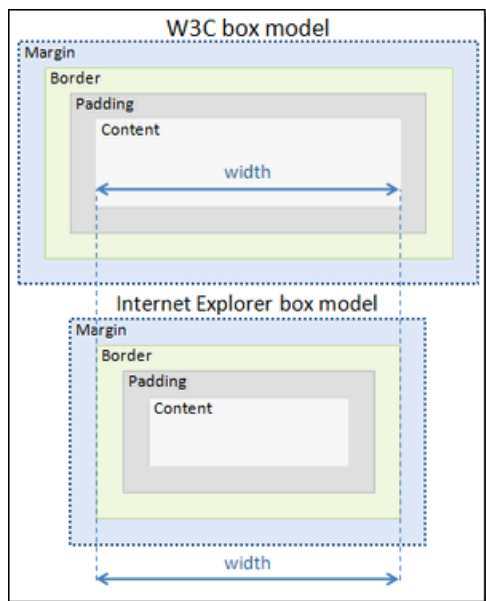


图 10-4 盒式模型对比



使用 `jQuery.support.boxModel` 属性来获取浏览器是否使用了 W3C 盒式模型。`true` 表示使用 W3C `boxModel`。

#### 4. 浮动样式

通过 JavaScript 脚本设置元素的 `float` 样式时, IE 和 FireFox 存在不同, IE 使用 `style.styleFloat`, FireFox 使用 `style.cssFloat`:

```
div.style.styleFloat = "left"; //IE
div.stlye.cssFloat = "left"; //FF
```

`jQuery.support.cssFloat` 属性返回 `true` 则表示可以使用 `cssFloat` 来设置 `float` 样式, IE 中返回 `false`。

另外, 可以通过 `CSS()` 方法设置 `float` 样式, jQuery 内部会自动帮助判断是使用 `styleFloat` 还是 `cssFloat`:

```
$("#divResult").css("float","left"); //兼容 IE 和 FF
```

因为 `support` 的属性众多, 无法一一讲解, 以下有一些非常棒的资源用于解释这些特性检测是如何工作的:

- ❑ <http://peter.michaux.ca/articles/feature-detection-state-of-the-art-browser-scripting>。
- ❑ [http://www.jibbering.com/faq/faq\\_notes/not\\_browser\\_detect.html](http://www.jibbering.com/faq/faq_notes/not_browser_detect.html)。
- ❑ <http://yura.thinkweb2.com/cft/>。

## 10.3 数组和对象操作

改变页面样式, 需要常常操作 DOM 对象或者 jQuery 包装集, 但是实现算法或者业务逻辑时往往操作的是数组和对象。

本节讲解最常用的数组和对象相关的工具函数。首先看一下 jQuery 中提供的和操作数组、对象有关的函数, 如表 10-4 所示。

表 10-4 操作数组和对象相关函数列表

函数签名	返回值	版 本	函数说明	参数说明
<code>jQuery.each(object, callback(indexInArray, valueOfElement))</code>	Object	1.0	通用例遍方法, 可用于例遍对象和数组。	<code>object</code> : 要遍历的数组或对象 <code>callback(indexInArray, valueOfElement)</code> : 遍历对象的函数。 <code>indexInArray</code> 表示当前遍历对象在数组中的索引, <code>valueOfElement</code> 表示当前遍历对象的值
<code>jQuery.extend( target, [ object1 ], [ objectN ] )</code>	Object	1.0	用一个或多个其他对象来扩展第一个对象, 返回被扩展的对象。	<code>target</code> : 待修改对象。 <code>object1- objectN</code> : 准备合并到 <code>target</code> 上的对象
<code>jQuery.extend( [ deep ], target, object1, [ objectN ] )</code>	Object	1.1.4	用一个或多个其他对象来扩展第一个对象, 返回被扩展的对象。	<code>deep</code> : 如果设为 <code>true</code> , 则递归合并。 <code>target</code> : 待修改对象。 <code>object1- objectN</code> : 准备合并到 <code>target</code> 上的对象

# 第 10 章 jQuery 工具函数

(续表)

函数签名	返回值	版 本	函数说明	参数说明
jQuery.grep( array, function(elementOfArray, indexInArray), [ invert ] )	Array	1.0	使用过滤函数过滤数组元素。	Array: 待过滤的对象 invert: 默认值 false, 表示函数返回数组中由过滤函数返回 true 的元素; 为 true 则返回过滤函数中返回 false 的元素集。 Function(elementOfArray, indexInArray): 此函数将处理数组每个元素。第一个参数为当前元素, 第二个参数为元素索引值。此函数应返回一个布尔值, 函数中的 this 表示全局的 window 对象
jQuery.makeArray(obj)	Array	1.2	将类数组对象转换为数组对象。 类数组对象有 length 属性, 其成员索引为 0 至 length - 1。实际中此函数在 jQuery 中将自动使用而无须特意转换	obj: 要被转成 JavaScript 原生数组的对象
jQuery.map(array, callback(elementOfArray, indexInArray))	Array	1.0	将一个数组中的元素转换到另一个数组中。 作为参数的转换函数会为每个数组元素调用, 而且会给这个转换函数传递一个表示被转换的元素作为参数。转换函数可以返回转换后的值、null (删除数组中的项目) 或一个包含值的数组, 并扩展至原始数组中	array: 待转换的数组 callback(elementOfArray, indexInArray): 转换函数, 第一个参数是当前遍历的元素, 第二个参数是元素索引值。此函数可返回任何值
jQuery.inArray(value, array)	Number	1.2	确定第一个参数在数组中的位置, 从 0 开始计数 (如果没有找到则返回-1)	value: 查询的值 array: 待查询的数组
jQuery.isArray(obj)	Boolean	1.3	判断对象是否是 Javascript 原生数组	obj: 待判断的对象
jQuery.merge(first, second)	Array	1.0	合并两个数组 返回的结果会修改第一个数组的内容——第一个数组的元素后面跟着第二个数组的元素。要去除重复项, 请使用\$.unique()	first: 待合并的第一个对象, 第二个对象会合并到第一个对象中。 Second: 待合并的第二个对象



(续表)

函数签名	返回值	版 本	函数说明	参数说明
jQuery.unique(array)	Array	1.1.3	删除数组中重复元素。 只处理删除 DOM 元素 数组，而不能处理字符 串或者数字数组。	array: DOM 数组
jQuery.parseJSON(json)	Object	1.4.1	接受一个 JSON 字 符串，返回解析后的对象	json: JSON 字符串

### 10.3.1 遍历数组和对象

jQuery 中提供了 jQuery 对象函数“.each”及工具函数“jQuery.each”都可以用来遍历对象，不同之处在于.each()函数用来遍历 jQuery 对象集合。

```
$("#img").each(function(i){  
    this.src = "test" + i + ".jpg";  
});
```

而 jQuery.each 可以用来遍历任何对象，包括 map 对象（包含属性的 Javascript 对象）或数组：

```
$.each([123, 456], function(index, value) {  
    alert(index + ': ' + value);  
});
```

输出结果为：

```
0: 123  
1: 456
```

jQuery.each 和.each()函数一样，都可以接受一个回调函数：

```
var map = {  
    "name": "ZZQ",  
    "age": "18"  
};  
$.each(map, function(key, value) {  
    alert(key + ': ' + value);  
});
```

输出结果为：  
name: ZZQ  
age: 18

调用回调函数时，会传入两个参数 indexInArray 和 valueOfElement。

indexInArray 是对象在集合中的索引值。当集合是数组时，比如上面的 “[52,97]”，indexInArray 表示数字索引，比如 0、1；当对象是 map 对象时，indexInArray 表示属性名称，比如 “name”、“age” 等。

valueOfElement 表示当前遍历的元素或属性的值。

### 10.3.2 过滤数组

使用 jQuery.each()函数可以遍历对象或数组，但是如果希望在遍历的同时改变集合中的元素，jQuery.each()函数就显得力不从心。jQuery.grep()函数也会遍历对象，是专门用于过滤



数组的，但其不能作用于 map 对象（动态从 map 对象中删除属性通常没有意义）。

jQuery.grep()函数中的过滤函数必须有一个返回值，默认情况下 true 表示保留此元素，false 表示删除此元素：

```
var arr = [ 1,2,3,4,5,6,7,8,9 ];
arr = jQuery.grep(arr, function(n, i){
    return (n != 5);
});
alert(arr.join(", "));
```

输出结果为：

```
1,2,3,4,6,7,8,9
```

上面使用 jQuery.grep()函数过滤掉了数组中值为“5”的元素，因为当遍历到 5 这个元素时，返回值“n != 5”为 false。

可以传递一个可选的 invert 参数，让过滤函数保留返回 false 的函数，去除返回 true 的函数（默认情况下正好相反）：

```
var arr = [ 1,2,3,4,5,6,7,8,9 ];
arr = jQuery.grep(arr, function(n, i){
    return (n != 5 );
}, true);
alert(arr.join(", "));
```

输出结果为：

```
5
```

发现结果正好和之前的相反。

过滤函数的签名是：

```
function(elementOfArray, indexInArray)
```

第一个参数 elementOfArray 表示当前数组中的对象，indexInArray 则表示对象在数组中的索引。

### 10.3.3 数组和对象合并

无论是数组或对象，都常常要遇到合并操作。传统的做法是使用循环进行数组或对象的遍历。在 jQuery 中提供了用于数组、对象合并的函数：

❑ jQuery.merge。

❑ jQuery.extend。

jQuery.merge 用于合并两个数组：

```
$.merge( [0,1,2], [2,3,4] )
```

结果为：

```
[0,1,2,2,3,4]
```

jQuery.merge()函数使用时有以下注意事项：

❑ 两个参数都只能是 JavaScript 原生数组。

❑ 第一个参数对象会被改变，第二个参数不变。





虽然 `jQuery.merge()` 函数返回的是合并后的数组对象，但是同时也会改变第一个参数。如果想保留第一个参数原始的状态，可以先将数组复制到另一份副本中：

```
var oldArrayClone = $.merge([], oldArray);
```

上面的代码使用 `oldArray` 复制了一个新的数组 “`oldArrayClone`”。

有了这个复制的副本，可以在合并操作时操作这个复制的数组，达到不改变原数组的目的：

```
var first = ['a','b','c'];  
var second = ['d','e','f'];  
var newArray = $.merge( $.merge([],first), second);
```

上面的代码运行后，`first` 和 `second` 两个数组都维持原样不变，`newArray` 则是两个数组合并后的新数组。

如果要合并的是两个 `map` 对象，则可以使用 `jQuery.extend()` 函数：

```
jQuery.extend( target, [ object1 ], [ objectN ] )  
jQuery.extend( [ deep ], target, object1, [ objectN ] )
```

`jQuery.extend()` 函数的签名看似有点复杂。首先是可选的 `deep` 参数，`deep` 参数表示是否执行递归复制（也称深层复制）。什么叫做递归复制呢？这主要是针对某个属性又是一个复杂的 `map` 对象而言的。比如下面的例子：

```
var object1 = {  
  apple: 0,  
  banana: {weight: 52, price: 100},  
  cherry: 97  
};  
var object2 = {  
  banana: {price: 200},  
  durian: 100  
};  
$.extend(object1, object2);
```

最后 `object1` 为：

```
{apple: 0, banana: {price: 200}, cherry: 97, durian: 100}
```

上例中省略了 `deep()` 函数，相当于使用了 `deep()` 的默认值 `false`。如果改成深层复制，结果就不一样了：

```
$.extend(true, object1, object2);
```

最后 `object1` 为：

```
object1 === {apple: 0, banana: {weight: 52, price: 200}, cherry: 97, durian: 100}
```

在执行对象复制的过程中，实际上是通过判断属性名称来决定是更新还是新增属性。更新属性的时候，`deep` 为 `false` 时，直接进行整个属性的替换。`deep` 为 `true` 的时候则会对属性的类型进行判断，如果属性也是一个 `map` 对象，则再次进行遍历。这个判断是一个递归过程，也就是说 `map` 对象的嵌套可以是多层次的。

`jQuery.extend()` 函数的 `target` 参数实际上有两个作用，当传递了 `target` 参数和至少一个 `object` 参数时，会将所有的 `object` 合并到 `target` 对象中：







## 第 10 章 jQuery 工具函数

```
var object = $.extend(object1, object2);
```

上例中，会合并 `object1` 和 `object2`，最后 `object` 和 `object1` 对象都是合并后的对象。如果在合并的时候不希望改变原对象，则可以传递一个空的 `target` 对象。

```
var object = $.extend({}, object1, object2);
```

这样只有 `object` 对象才是合并后的对象，`object1` 对象不改变。

使用 `jQuery.extend()` 函数可以扩展 `jQuery` 对象。如果 `jQuery.extend()` 只传递一个对象，则表示扩展的是 `jQuery` 对象：

```
var object1 = {  
    myMethod: function() { alert("my method"); },  
    myName: "zzq"  
};  
jQuery.extend(object1);  
$.myMethod();  
alert($.myName);
```

上面的例子为 `jQuery` 对象扩展了 `myMethod()` 方法和 `myName` 属性。

需要注意，`jQuery.extend` 不能用于合并数组。因为此方法是通过属性索引访问的，将索引相同的项合并。数组的属性索引是数字 0、1、2 等，这会导致如下的结果：

```
var first = ['a','b','c'];  
var second = ['d','e','f'];  
$.extend(first, second);
```

最后 `first` 为：

```
['d','e','f'];
```

因为 `d` 和 `a` 的索引都是 0，所以会用 `d` 覆盖 `a`。

### 10.3.4 数组和对象转换

JavaScript 中常用的集合有数组、`map` 对象（包含了属性的 JavaScript 对象）、类数组对象。类数组对象是指类似 `jQuery` 对象的对象，比如使用 `$()` 获取到的对象，拥有 `length` 属性，并且可以通过属性访问器 `[]` 访问，但是并不是真正的 JavaScript 的数组对象，比如没有数组对象的 `pop()`、`reverse()` 等方法。

下面举例说明 `jQuery` 中的各种集合转换函数。

将类数组对象转化为数组对象：

```
var obj = $('li');  
var arr = $.makeArray(obj); 输出结果为：
```

`obj` 是一个 `jQuery` 对象：

```
(typeof obj === 'object' && obj.jquery) === true;
```

`arr` 是使用 `makeArray()` 函数获取到的数组对象：

```
jQuery.isArray(arr) === true;
```

`jQuery.isArray()` 函数可以判断对象是否是 JavaScript 原生数组对象。

在过滤函数中讲解了使用 `jQuery.grep()` 函数过滤数组，还有一个既可以过滤也可以用于





数组转换的函数 `jQuery.map()`。下面的例子将数组中的每一个元素加上自身的索引值：

```
$.map( [0,1,2], function(n, i){  
    return n + i;  
});
```

输出结果为：

```
0,2,4
```

`jQuery.map()`的回调函数和 `jQuery.grep()`一样，第一个参数是数组中的对象，第二个参数是对象在集合中的索引。

`jQuery.map()`中的回调函数返回的是数组元素的最终状态值，如果返回 `null` 则表示删除当前元素。下面的例子将删除大于 3 的元素：

```
$.map( [1,3,5,7], function(n){  
    return n > 3? n : null;  
});
```

最后的结果为：

```
1,3
```

`jQuery.map()`函数基本上可以替代 `jQuery.grep()`函数。

使用 `jQuery.map()`函数有两点需要注意：

- ☐ `jQuery.map()`函数可以作用在类数组对象中。
- ☐ `jQuery.map()`函数返回的是数组对象，如果遍历的是类数组对象，则会转换成数组对象。

### 10.3.5 排序和过滤 DOM 元素集合

`jQuery 1.4.1` 版本中添加了 `jQuery.unique()`函数，作用是排序和过滤 DOM 集合。`jQuery.unique()`函数会将 DOM 集合中重复的 DOM 对象除去，并且按照在 HTML 中出现的顺序排序。

假设页面上有如下元素：

```
<div id="one">one</div>  
<div id="two">two</div>  
<div id="three">three</div>
```

使用下面的语句：

```
var divs = $("#three").get();  
divs = divs.concat($("#two")[0]);  
divs = divs.concat($("#one")[0]);  
divs = divs.concat($("#two")[0]);  
thisPage.console("过滤前的集合：");  
$(divs).each(function() { thisPage.console(this.id); });  
jQuery.unique(divs);  
thisPage.console("过滤后的集合：");  
$(divs).each(function() { thisPage.console(this.id); });
```

输出结果为：

```
过滤前的集合：  
three
```



```
two
one
two
过滤后的集合:
one
two
three
```

上例中，首先使用 jQuery 选择器选中了 ID 为 three 的 div，使用 get() 函数将返回一个 DOM 对象数组而不是 jQuery 对象。对于数组，使用原始的 concat 进行连接。输出时，three 排在第一个所以最先输出。因为添加了两两 two 元素，所以集合中会存在两个 two 元素。

当对这个 DOM 数组使用了 jQuery.unique() 函数后，发现 two 元素被过滤为只选择一个，并且三个 div 按照 HTML 中出现的顺序进行了排序。

jQuery.unique() 函数在 jQuery 内部也会被使用，在 1.4 版本的 jQuery 中，使用选择器返回的对象都是按照它们在 HTML 中出现的顺序排序的，比如：

```
$("#three, #two, #one").each(function() { thisPage.console(this.id); });
```

虽然选择器表达式中，ID 选择器的顺序是 three、two、one，但是最后的结果依然是 one、two、three。

### 10.3.6 转换 JSON 字符串

使用 JavaScript 原生的 eval() 函数可以将 JSON 字符串转换为对象，jQuery 在 1.4 版本中加入了 jQuery.parseJSON() 函数，也是将一个标准的 JSON 字符串转化为 JavaScript 对象并返回：

```
var obj = jQuery.parseJSON('{ "name": "zzq" }');
alert( obj.name === "zzq" );
```

jQuery.parseJSON() 和 eval 是不同的。如果 JSON 字符串参数是空字符串、null、undefined 或者不传递 JSON 字符串时，jQuery.parseJSON() 返回 null：

```
thisPage.console("parseJSON:");
thisPage.console($.parseJSON());
thisPage.console($.parseJSON(""));
thisPage.console($.parseJSON(null));
thisPage.console($.parseJSON(undefined));
```

输出结果：

```
parseJSON:
null
null
null
null
```

而如果是 eval 则会出现异常，因为构建了不正确的 JavaScript 语句。

另外在 jQuery.parseJSON() 内部，会首先判断是否存在 window.JSON.parse() 函数，如果存在则使用此函数转换。下面是 jQuery.parseJSON() 函数源代码片段：

```
return window.JSON && window.JSON.parse ?
window.JSON.parse( data ) :
(new Function("return " + data))();
```

有关 JSON 格式字符串的细节，可以参考：



<http://json.org/>

然而不幸的是, jQuery 仍然没有提供将一个对象转换为 JSON 字符串的函数。需要开发人员自己扩展。在本书中的下列文件中:

【代码路径: jQueryStorm.Web/Static/Common/js/Core.js】

里面提供了一个工具函数 toJSON()。首先来看一下 toJSON() 函数的使用:

```
var obj =  
    {  
        com: "elong",  
        WebAddress: "www.elong.com"  
    };  
thisPage.console(jQuery.toJSON(obj));
```

输出结果为:

```
{"com": "elong", "WebAddress": "www.elong.com"}
```

toJSON() 函数能够把对象序列化成 JSON 格式的字符串。toJSON() 函数还有一个两个参数的重载:

```
toJSON(obj, compact)
```

compact 是一个可选参数, 默认为 false。如果传递为 true, 则返回的结果字符串会在每个属性前添加一个空格。

## 10.4 其他工具函数

除了上面介绍的工具函数, jQuery 还有很多实用的工具函数, 下面将一一进行介绍。

### 10.4.1 字符串 trim 操作

很不巧, JavaScript 没有去掉字符串头尾空格的函数, 虽然大部分语言中都由 trim() 函数来实现此功能。

jQuery 提供了 jQuery.trim() 函数用来实现过滤头尾的空格:

```
var word = "    Hello    ";  
word = jQuery.trim(word);
```

word 中的前后空格都会被去掉。

### 10.4.2 判断函数

jQuery 工具函数中, 包括下列用于判断的函数:

- ❑ jQuery.isEmptyObject( object )。
- ❑ jQuery.isFunction( obj )。
- ❑ jQuery.isPlainObject( object )。
- ❑ jQuery.isXMLDoc( node )。
- ❑ jQuery.isArray( obj )。

jQuery.isEmptyObject() 函数用来判断一个对象是否是一个空对象, 即没有属性的对象:





```
jQuery.isEmptyObject({}) //返回: true  
jQuery.isEmptyObject({ foo: "bar" }) //返回: false
```

jQuery.isFunction()函数用来判断一个对象是否是 function。

```
jQuery.isFunction(function({})); //返回: true  
jQuery.isFunction("abc"); // 返回: false
```

jQuery.isPlainObject()函数用来检查一个对象是否是 JavaScript 中的对象（用“{}”或者“new object”创建的对象）：

```
jQuery.isPlainObject({}) //返回: true  
jQuery.isPlainObject("test") //返回: false
```

jQuery.isXMLDoc()函数检测一个 DOM 结点是否在 XML 文档结点中（或者自身是否是 XML 结点）：

```
jQuery.isXMLDoc(document) //返回: false  
jQuery.isXMLDoc(document.body) //返回: false
```

jQuery.isArray()函数判断一个数组是否是原生的 JavaScript 数组，而不是类数组。在本章“数组和对象转换”一节使用过：

```
$.isArray([]); //返回: true  
$.isArray($(document)); //返回: false
```

### 10.4.3 jQuery 中的全局 eval 函数

jQuery.globalEval()的作用和 JavaScript 原生的 eval()函数一样，能够动态地执行脚本语句。但是 jQuery.globalEval 是全局 eval 函数，所谓“全局”是指函数的上下文是 Window 对象：

```
function test() {  
    jQuery.globalEval("var obj1 = true;");  
    eval("var obj2 = true");  
};  
test();  
alert(obj1);  
alert(obj2);
```

上面的实例中，obj1 是使用 jQuery.globalEval()函数创建的，相当于在 Window 上创建了 obj1 对象，所以可以使用 alert 语句访问，结果为 true。而 obj2()函数使用 eval 创建，作用域仅限 test()函数体内，所以再次访问时提示语句出错，obj2 对象未定义。

### 10.4.4 制造一个空函数

空函数通常没有任何功能，但是在开发插件的时候，有时候回调函数是一个可选的参数，而插件内部要调用回调函数，如果没有传递任何函数则会引发空引用异常，所以需要创建一个空函数。再或者在本书第 13 章中，实现的 class.create()方法内部就使用了 jQuery.noop 创建一个空函数。

下面演示在创建一个空函数作为 xmlhttprequest 对象时的回调方法：

```
xhr.onreadystatechange = jQuery.noop;
```





### 10.4.5 检查结点包含关系

检查一个结点是否包含另外一个结点，可以使用工具函数：

```
jQuery.contains( container, contained )
```

这是 jQuery 1.4 版本加入的函数。两个参数都是 DOM 元素，如果第一个 DOM 元素包含第二个 DOM 元素，则返回 true：

```
jQuery.contains(document.documentElement, document.body); // true
jQuery.contains(document.body, document.documentElement); // false
```

### 10.4.6 修改函数上下文

JavaScript 中的 this 指向“函数的调用者”，如果忘记了这部分知识，可以在第 2 章中回顾。正因为 this 的这种可变性，为编写程序制造了麻烦，比如，下面声明一个 myClass 对象，并且有函数 checkName() 和 test() 函数：

```
var myClass =
{
    checkName: function() { return true; },
    test: function() {
        if (this.checkName()) { //在 test()函数中通过“this”又调用了 checkName()函数
            alert("ZZQ");
        }
    }
}
```

编写 JavaScript 时常常使用上例中的面向对象的方式，将方法或属性都包装成一个类。在 C# 等一些高级语言中，上面的例子是没有错误的，因为 this 始终指向当前实例。但是在 JavaScript 中则不同。如果将 test() 函数用于事件绑定：

```
$("#btnTest").click(myClass.test);
```

当单击 btnTest 按钮时，会提示找不到 checkName() 函数的错误：

```
this.checkName is not a function
```

这是因为 this 指针已经不是指向 myClass 类，而是事件的调用者，即 DOM 对象“btnTest”，此 DOM 对象上并没有 checkName() 方法。

要解决这一问题，需要使用 jQuery.proxy() 函数，如表 10-5 所示。

表 10-5 jQuery.proxy() 函数重载列表

函数签名	返回值	加入版本	参数说明
jQuery.proxy( function, context )	Function	1.4	function: 要改变上下文的函数 context: 函数上下文，返回的 function 对象中 this 将始终表示此对象
jQuery.proxy( context, name )	Function	1.4	context: 函数上下文，返回的 function 对象中 this 将始终表示此对象 name: context 对象中要返回的 function 对象的函数名称

下面使用 jQuery.proxy() 函数来解决 this 指针的引用问题：

```
$("#btnTest").click(jQuery.proxy(myClass.test, myClass)); //第一种用法
```





```
$("#btnTest").click(jQuery.proxy(myClass,"test"));
```

//第二种用法

jQuery.proxy()会返回一个 function 对象，即返回一个函数，这个函数内的 this 始终指向指定的 context 对象。上面将 context 指定为 myClass，所以单击事件触发时，调用 this.checkName()函数实际上是调用 myClass.checkName()函数。

jQuery.proxy()的实现原理实际上是利用了 JavaScript 原生的 apply()函数，下面是 jQuery.proxy()的一段核心代码：

```
return fn.apply( thisObject || this, arguments );
```

其中 fn 是要调用的函数，thisObject 是函数上下文，arguments 是函数的参数。

虽然 jQuery.proxy()能够解决此问题，但因为这是一个工具函数，无法使用链式操作，所以看起来不够优雅。下面使用一种更加优雅的方式。为 JavaScript 的 Function 添加原型：

```
Function.prototype.proxy = function(context) {  
    return jQuery.proxy(this, context);  
};
```

现在，用一种更加优雅的语法，实现制定函数上下文的事件绑定：

```
$("#btnTest").click(myClass.test.proxy(myClass));
```

这种方式的诀窍在于为函数添加原型函数 proxy()，在此函数中，this 是指向函数的调用者，也就是 Function 本身，在内部还是调用 jQuery.proxy()实现。

### 10.4.7 jQuery 中的队列函数

队列相关的函数也全都属于工具函数。有关队列函数的详细介绍请参见“jQuery 动画”一章中的“jQuery 队列”节的内容。

## 10.5 扩展 jQuery 工具函数

虽然 jQuery 中提供了大部分常用的脚本功能，但是依然不能满足所有的需求。所以常常要对工具函数做一下扩展，添加一些自定义的工具函数。比如在讲解 jQuery.parseJSON()函数时，为了添加一个将对象转换成 JSON 串的方法，编写了一个自定义 jQuery.toJSON()函数。

通常扩展 jQuery 工具函数有两种方式，下面分别介绍。

### 10.5.1 使用 JavaScript 扩展工具函数

借助 JavaScript 的动态语言的特性，可以随时为 jQuery 对象添加静态工具函数：

```
jQuery.myMethod = function()  
{  
    alert("myMethod");  
}  
jQuery.myMethod();
```

上面的代码添加了 jQuery.myMethod()函数。





### 10.5.2 使用 jQuery.extend() 函数扩展工具函数

jQuery.extend() 函数也可以用来扩展静态的 jQuery 工具函数：

```
jQuery.extend({ myMethod: function() {  
    alert("myMethod");  
}  
});  
jQuery.myMethod();
```

在 jQuery.extend() 函数中，当值传递一个对象时，默认的行为是扩展 jQuery 对象本身。

通常称为“工具函数”的方法都是静态的。但是在 jQuery 工具函数中，仍然包括一些非静态的实例函数，比如队列相关的函数，不过这只是极少的一部分。大部分实例函数都是作为 jQuery 插件的形式提供的。在第 11 章中将对 jQuery 插件做详细的讲解。

## 10.6 小结

本章介绍了 jQuery 的工具函数，在 API 中分类为 Utilities 的函数。

工具函数能够帮助开发人员完成许多与业务逻辑无关的任务，是最基础的函数。浏览器相关函数是开发跨浏览器的插件所必须掌握的，数组和对象相关函数则能够帮助开发人员快速地完成对象和数组的各种复杂操作，熟练地使用工具函数能够提高开发效率。

一个成型的系统和前段框架，往往已经实现了自己的脚本类库。在 jQuery 的工具函数库的基础上，加入自己系统的工具函数扩展，就可以轻而易举地打造一个高复用的、精简的脚本类库。但要切记不要重复开发 jQuery 已有的功能，这样不仅会提高使用脚本框架的人的学习成本，而且会造成类库代码的膨胀。







## 第 11 章



# 拿来主义——jQuery 插件

jQuery 丰富的插件一直是 jQuery 类库最受欢迎的原因之一。目前已经存在的 jQuery 插件几乎包括了能想到的任何功能。即使 JavaScript 技术水平一般的开发人员，也可以通过插件快速地向页面实现酷炫的效果。虽然 jQuery 插件的优势十分明显，但是对于大中型网站来说，还是要尽量开发自己的轻量级控件。因为 jQuery 插件往往占用的空间较大，并且难以二次开发。在第 13 章将会讲解如何使用 jQuery 开发脚本框架。

本章将主要讲解 jQuery 插件的开发和使用，并列举了表单验证插件和自动完成插件这两个插件例子。



## 11.1 jQuery 插件基础

### 11.1.1 jQuery 插件介绍

jQuery 插件通常是一个实例方法，能够在页面上快速地实现某些复杂的功能。下面以被评为 2009 年最佳插件之一的密码强度校验插件为例，了解何为“插件”：

【代码路径：jQueryStorm.Web/chapter11/ Demo-PasswordStrength.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery - 密码强度检测插件</title>
  <script src="../../Static/common/js/jquery-1.4.2.min.js"
type="text/javascript"></script>
  <script src="../../Static/common/js/jquery.pwdstr-1.0.source.js"
type="text/javascript"></script>
</head>
<body>
  <div id="divMsg">
    [输出窗口]</div>
  <div>
    <input type="text" id="txtPassword" />
  </div>

  <script type="text/javascript">
    $(function() {
      $("#txtPassword").pwdstr('#divMsg'); //对 id 为 txtPassword 的元素应用插件
    });
  </script>
</body>
</html>
```

输出效果如图 11-1 所示。

在输入框中输入密码，会在上面的 div 中显示密码被穷举破解需要的时间。

这是一个典型的插件引用，插件的使用通常很简单，上例中，首先需要引用插件的脚本代码：

```
<script src="../../Static/common/js/jquery.pwdstr-1.0.source.js"
type="text/javascript"></script>
```

应用插件，通常只需要一句脚本：

```
$("#txtPassword").pwdstr('#divMsg');
```

上面的代码表示要对 ID 为 txtPassword 的 input 控件进行密码强度校验，将结果输出在 ID 为 divMsg 的图层中。

图 11-1 密码强度校验插件

### 11.1.2 区别 jQuery 插件与工具函数

jQuery 插件和 jQuery 工具函数很相似，都是对功能的封装。但是不同之处在于 jQuery 工具函数通常比较简单，并且大部分都是静态函数，都是通过 jQuery 或“\$”变量引用的。而 jQuery 插件通常通过扩展 jQuery 对象，应用在 DOM 元素上。比如上例中的密码强度验



证插件，就是先使用 jQuery 选择器，返回 jQuery 对象，然后执行扩展的 `pwdstr()` 函数。

### 11.1.3 寻找合适的 jQuery 插件

在 jQuery 的官方网站上，收集了几乎所有的 jQuery 插件，并且按照分类进行组织：jQuery 插件官方网站如图 11-2 所示，网址是 <http://plugins.jquery.com/>。

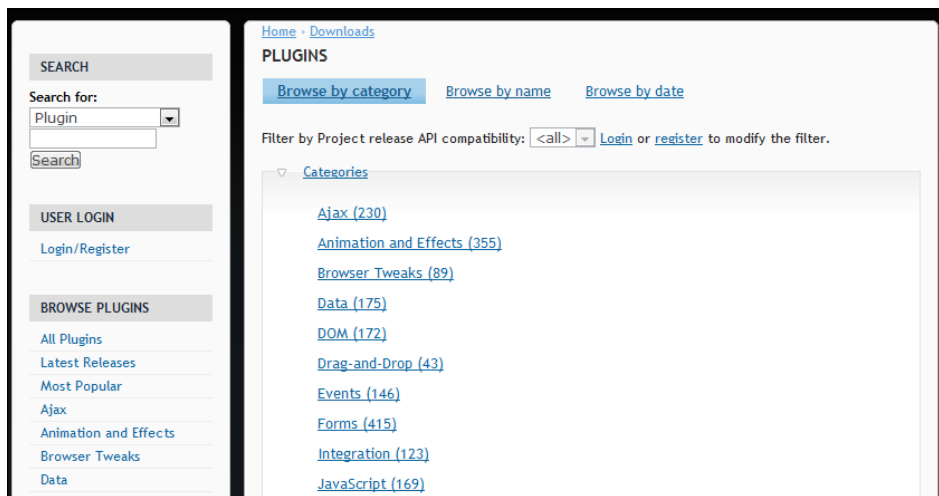


图 11-2 jQuery 插件官方网站截图

官方网址有一篇很好的文档描述了如何做插件开发：<http://docs.jquery.com/Plugins/Authoring>。当然通过搜索引擎寻找好用的插件，也是不错的选择。

### 11.1.4 合理使用 jQuery 插件

虽然使用 jQuery 插件可以实现很多的功能，但是往往一个复杂的插件包括图片、样式和脚本库，会让页面增加很多的负担。而且因为过分依赖插件简单的实现，不了解插件内部的实现原理，当出现 bug 时进行修复需要花费更多的时间。

所以，对于大型网站和页面流量较大的站点，建议在 jQuery 类库上自己实现 jQuery 插件，比如将全站的弹出框、日历框等进行封装。

对于中小网站或流量较小的站点，比如公司展示型网站，为了快速开发酷炫的页面，可以考虑使用第三方的 jQuery 插件。

在第 12 章中将讲解 jQuery UI，这是一个 jQuery 官方提供的，用于 UI 的插件。UI 插件方面应首选使用 jQuery UI 或者基于 jQuery UI 的二次开发。

接下来的内容里将讲解如何开发 jQuery 插件。

## 11.2 jQuery 插件开发

本节以前面密码强度检测插件为例，来讲解 jQuery 插件开发的流程和注意事项。





### 11.2.1 为插件起一个名字

首先要根据自己插件的功能，为插件起一个不会和其他插件或函数重复的名字。

比如密码强度插件的名字叫做“pwdstr”。就命名而言这并不是一个好名字，但是不会重名。

### 11.2.2 编写结构代码

现在可以编写插件代码。jQuery 插件通常都是为 jQuery.fn 对象做扩展。jQuery.fn 就是使用 jQuery 选择器选择后的 jQuery 对象。

```
jQuery.fn.pwdstr = function() {  
    //逻辑代码  
};
```

如果插件不是简单的一个函数，可以使用特殊的 jQuery.fn.extend() 函数。在学习 jQuery.extend 时知道使用 jQuery.extend 可以扩展 jQuery 工具函数或者合并对象。jQuery.fn.extend() 函数则可以用来扩展 jQuery 对象：

```
jQuery.fn.extend({  
    check      : function() { ... },  
    uncheck    : function() { ... },  
    toggleCheck : function() { ... }  
});
```

如果在控件开发中，还希望使用“\$”快速引用 jQuery，但是页面上还引用了其他类库，存在“\$”可能冲突的问题，有一种方式可以解决：

```
function($) {  
    $.fn.pwdstr = function() {  
        //逻辑代码  
    };  
}(jQuery);
```

上面的代码相当于建立了一个作用域，在 function 的作用域中，“\$”代表 jQuery 对象。除非需要，否则不建议这么做，仅仅将这种方式作为一种代码混乱后的补救措施。

当然，也可以把作用域的技巧和 jQuery.fn.extend() 函数结合起来使用，在 pwdstr 插件中就是这么使用的：

```
(function($){  
    $.fn.extend({  
        pwdstr: function(el) {  
            //实现部分  
        }  
    });  
})(jQuery);
```

### 11.2.3 设计插件参数

基本上每一个插件都要传递一个 options 参数，这是一个 map 对象，里面包含多个属性。使用这种方式可以让插件的签名变得简单。pwdstr 插件因为只需要一个参数，所以没有 options 参数。下面是一个标准的插件 options 参数的例子。

```
jQuery.fn.pluginName = function( options ) {
```



```
settings = jQuery.extend({  
    name: "defaultName",  
    size: 5,  
    global: true  
}, options);  
}
```

options 参数的值通常都是可选的，也就是可以不传递，在函数内部会设置一个默认值。比如上面的例子中，name 默认值就是 defaultName，如果传递了新的 name 则会覆盖默认值。使用 jQuery.extend() 函数可以轻易地将传递的参数和默认值进行合并。

接下来在插件内部，就可以使用这些参数了。

#### 11.2.4 使用插件

在完成了插件的代码实现部分后，首先要将这部分代码放到一个地方，并在页面上引用。通常将插件的代码放到单独的脚本文件中，起名为“jQuery.插件名称.js”，比如 pwdstr 插件最后放在了 jQuery.pwdstr.js 文件中。

在页面上引用该文件后，就可以在页面上使用此插件了：

```
$("#txtPassword").pwdstr('#divMsg');
```

#### 11.2.5 插件开发要点

通过上面的流程，已经知道了一个完整的插件是如何开发的。下面是一些 jQuery 插件开发的要点：

- ❑ 将插件命名为“jQuery.插件名称.js”，比如：jQuery.myPlugin.js。
- ❑ 将插件函数绑定到 jQuery.fn 对象上，将工具函数绑定到 jQuery 对象上。
- ❑ 在函数内部，this 指向到当前的 jQuery 对象上。
- ❑ 所有的方法或对象都要以“；”结尾，避免在压缩代码时出错。
- ❑ 插件函数最后返回 jQuery 对象，以便支持链式操作。
- ❑ 使用 this.each 遍历 jQuery 对象集合。
- ❑ 尽量使用 jQuery 而不是别名“\$”。

本节介绍了如何开发 jQuery 插件。接下来将介绍几个比较实用的插件。

### 11.3 实战表单验证插件

在提交表单前常要对用户的输入进行校验。ASP.NET 的验证控件就是用于此目的，可以同时进行客户端和服务端验证。但是验证控件的使用具有局限性，而且在 MVC 项目中经常使用自己的客户端验证框架。

jQuery.validate 插件是颇为流行的一套验证插件，即将被 ASP.NET MVC 的下一个 3.0 版本所集成。目前当创建一个 ASP.NET MVC 2 的项目时，默认就会引入 jQuery.validate 插件。

插件首页：<http://bassistance.de/jquery-plugins/jquery-plugin-validation/>。

插件文档：<http://docs.jquery.com/Plugins/Validation>。



配置说明：<http://docs.jquery.com/Plugins/Validation/validate#options>。

### 11.3.1 应用实例

表单验证插件的效果如图 11-3 所示。

图 11-3 表单验证插件效果

在这个表单上，有姓名、E-mail、网址、内容这几个需要用户填充的字段。其中姓名、E-mail 和内容是必填项。同时各个字段的验证都拥有自己的验证规则。如果用户输入不满足规则，页面上会给出相应的提示信息。

本实例是使用 validation 插件实现的。下面是完整的实例代码：

```
【代码路径：jQueryStorm.Web/chapter11/ Demo-validation.aspx】
<% @ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>jQuery PlugIn - 表单验证插件实例 Validate </title>
    <script src="../../Static/common/js/jquery-1.4.2.min.js"
type="text/javascript"></script>
    <script src="../../Static/common/js/jquery.validate/jquery.validate.min.js"
type="text/javascript"></script>
    <script src="../../Static/common/js/jquery.validate/localization/messages_cn.js"
type="text/javascript"></script>
    <style type="text/css">
        body
        {
            font-size:12px;
        }
        /* form 中显示文字的 label */
        .slabel
        {
            width:100px;
            display: -moz-inline-box;
            line-height: 1.8;
            display: inline-block;
            text-align:right;
        }
        /* 出错样式 */
        input.error, textarea.error
```

```

    {
        border: solid 1px #CD0A0A;
    }
    label.error
    {
        color:#CD0A0A;
        margin-left:5px;
    }
    /* 深红色文字 */
    .textred
    {
        color:#CD0A0A;
    }
</style>
</head>
<body>
<form id="commentForm" method="get" action="">
<fieldset style="width:500px;"><legend>表单验证</legend>
    <p><label for="cname" class="slabel"><em class="textred">*</em> 姓名:</label>
        <input id="cname" name="name" size="25" class="required" minlength="2" />
    </p>
    <p><label for="cemail" class="slabel"><em class="textred">*</em> E-Mail:</label>
        <input id="cemail" name="email" size="25"/>
    </p>
    <p><label for="curl" class="slabel">网址:</label>
        <input id="curl" name="url" size="25" class="url" value="" />
    </p>
    <p><label for="ccomment" class="slabel"><em class="textred">*</em> 内容:</label>
        <textarea rows="2" id="ccomment" name="comment" cols="20" class="required"
style="height:80px;"></textarea>
    </p>
    <p style="text-align:center;">
        <input class="submit" type="submit" value="提交" />
    </p>
</fieldset>
</form>
<script type="text/javascript">
    /*用户自定义方法*/

    /*事件绑定*/
    $(function()
    {

    });

    /*加载时执行的语句*/
    $(function()
    {
        $("#commentForm").validate(
        {
            errorClass: "error",
            submitHandler: function(form)
            {
                //如果想提交表单, 需要使用 form.submit()而不要使用$(form).submit()
                alert("submitted!");
            },
            rules: {
                //为 name 为 email 的控件添加两个验证方法 required()和 email()
                email: { required: true, email: true }
            },
            messages: {
                //为 name 为 email 的控件的 required()和 email()验证方法设置验证失败的消息内容

```





email: {required:"需要输入电子邮箱",email:"电子邮箱格式不正确"}

本实例在页面加载时，对 ID 为“commentForm”的 form 元素应用了“validate”插件。在调用 validate() 函数时，传递了 errorClass、rules 和 messages 这几个属性。这几个属性仅仅设置了 email 字段的录入规则。E-mail 是使用编写脚本的方式添加的验证规则，而其他字段并没有编写相应的代码，而是通过应用 CSS 样式类的形式添加的验证功能。接下来将详细讲解 validate 插件。

### 11.3.2 验证方法

验证方法是验证某一个控件是否满足某些规则的方法,返回一个 `boolean` 值。比如 `email()` 方法验证内容是否符合 E-mail 格式,符合则返回 `true`。下面是 `validate` 插件内置的 `email` 验证方法的源代码:

```
email: function(value, element) {
    // contributed by Scott Gonzalez:
    //http://projects.scottsplayground.com/email_address_validation/
    return this.optional(element) ||
```

`email()`函数接受两参数，`value` 和 `element`。`value` 表示元素的输入值，`element` 表示触发验证的表单元素。`email` 函数首先使用 `optional` 方法检测元素是否合法，然后使用一个正则表达式检查输入的内容是否满足 `email` 格式验证，并且使用 `test` 方法，返回验证结果 `true` 或者 `false`。

关于 validation 插件的说明, 请参考官方文档: <http://docs.jquery.com/Plugins/Validation>。

在文档的“List of built-in Validation methods”一节中列出了所有内置的验证方法，同时插件还提供了 additional-methods.js 文件，里面包含了更多的验证方法，引入后即可启用。



### 11.3.3 验证消息



验证消息就是验证方法失败后显示的文字内容。验证消息一定关联在某一个验证方法上，并且全局的验证消息保存在 `jQuery.validator.messages` 属性中。默认的 `validate` 类库自带英文验证消息：



```
messages: {
    required: "This field is required.",
    ...
});
```

上面说明当 `required` 验证方法验证失败时显示 “This field is required.” 这条消息，在下载文件的 `localization` 文件夹中，包含了各国语言的基本验证消息，如同本实例一样引入不同的语言文件即可实现语言切换：

```
<script src="../../Static/common/js/jquery.validate/localization/messages_cn.js"
type="text/javascript"></script>
```

语言文件的内容举例：

```
jQuery.extend(jQuery.validator.messages, {
    required: "必选字段",
    ...
});
```

现在必填项的问题提示就变成了中文。

除了全局默认的验证消息，也可以为某一个表单元素设置特有的验证消息，比如本文实例中，为 `email` 元素设置了特有的验证消息：

```
messages: {
    //为 name 为 email 的控件的 required()和 email()验证方法设置验证失败的消息内容
    email: {required: "需要输入电子邮箱", email: "电子邮箱格式不正确"}
```

`options` 的 `messages` 属性可以针对某一个表单元素设置验证消息，第一个 `email` 表示 `email` 元素，值是一个集合。`required` 就表示 `required()` 验证函数。第二个 `email` 表示是 `email()` 验证函数。

### 11.3.4 验证规则

验证规则就是这样的语意语句：在元素 A 上，使用验证方法 A 和验证方法 B 进行验证。

验证规则将元素与验证方法关联起来，因为验证方法同时也关联了验证消息，所以元素与消息也关联了起来。

为一个元素添加验证规则有多种方式。本实例的“姓名”元素使用了 CSS 样式规则和元素属性规则：

```
<input id="cname" name="name" size="25" class="required" minlength="2" />
```

`class` 元素属性设置元素的 CSS 样式类，因为样式类中添加了 `required` 类，所以会和 `required()` 验证函数关联。这种规则叫做 CSS 样式规则。

`minlength` 元素属性也会自动和 `minlength()` 验证函数关联，这种规则叫做元素属性规则。另外还可以通过编程的方式进行关联：

```
rules: {
    //为 name 为 email 的控件添加两个验证方法:required()和 email()
    email: { required: true, email: true }
},
```

上面的语句表明为 `email` 表单对象添加了 `required()` 和 `email()` 验证函数。





### 11.3.5 表单提交

默认情况下，当验证函数失败时表单不会提交。但是可以通过添加 `class="cancel"` 的方式让提交按钮跳过验证：

```
<input type="submit" class="cancel" name="cancel" value="Cancel" />
```

当表单提交时，会触发 `options` 中 `submitHandler` 属性设置的函数。

```
submitHandler: function(form)
{
    //如果想提交表单，需要使用 form.submit()而不需要使用$(form).submit()
    alert("submitted!");
},
```

此函数的签名同上。可以在这个函数中，编写在表单提交前需要处理的业务逻辑。

需要注意当最后以编程的方式提交表单时，一定不要使用 `jQuery` 对象的 `submit()` 方法。因为此方法会触发表单验证，并且再次调用 `submitHandler()` 设置的函数，会导致递归调用。

此函数的参数 `form` 就是表单对象，用途就是不进行验证提交表单：

```
form.submit()
```

### 11.3.6 DEBUG 模式

在开发阶段我们通常不希望表单被真正提交，虽然可以通过本实例中重写 `submitHandler()` 函数来实现。但是还有更好的方式，可以在 `submitHandler()` 函数内完成正式提交的逻辑，然后通过设置 `options` 的 `debug` 属性，来达到即使验证通过也不会提交表单的目的：

```
$(".selector").validate({
    debug: true
})
```

### 11.3.7 多表单验证

有时会在一个页面上出现多个 `form`，因为 `validate` 控件是针对 `form` 对象进行包装的，所以可以控制哪些 `form` 对象需要验证。同时为了方便一次设置页面上所有的应用了 `validate` 控件的 `form` 对象，提供了 `jQuery.validator.setDefaults()` 函数让我们可以一次设置所有的默认值。

```
jQuery.validator.setDefaults({
    debug: true
});
```

## 11.4 实战自动完成插件

目前在 `jQuery UI` 中，也提供了一个用于自动完成的插件，在第 12 章 `jQuery UI` 中将做详细讲解。本节介绍的 `autocomplete` 插件是在 `jQuery UI` 出现中自动完成控件出现之前就发布的，并且功能较 `jQuery UI` 中的要更灵活更强大一些。正所谓白菜萝卜各有所爱，如何取舍就要由开发人员自己决定了。



autocomplete 插件能够实现类似于 Google Suggest 的效果，如图 11-4 所示。

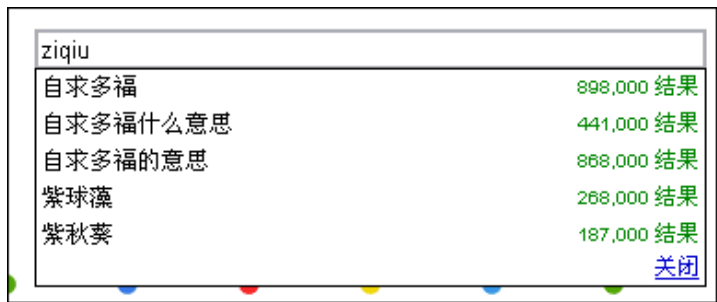


图 11-4 Google Suggest 效果

插件首页:

<http://bassistance.de/jquery-plugins/jquery-plugin-autocomplete/>。

插件文档:

<http://docs.jquery.com/Plugins/Autocomplete>。

配置说明:

<http://docs.jquery.com/Plugins/Autocomplete/autocomplete#options>。

#### 11.4.1 应用实例

本实例演示的是使用 autocomplete 插件完成对输入城市的自动提示效果，如图 11-5 所示。

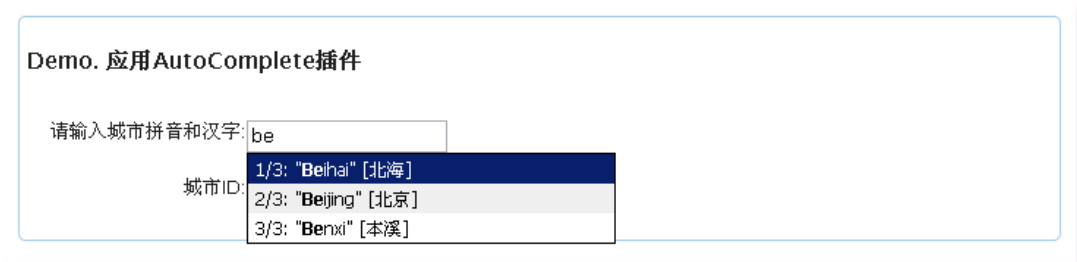


图 11-5 自动完成插件效果

实例代码:

```
【代码路径: jQueryStorm.Web/chapter11/ Demo-autocomplete.aspx】
<% @ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>jQuery PlugIn - 自动完成插件实例 AutoComplete </title>
    <link href="~/Static/common/js/jquery.autocomplete/jquery.autocomplete.css"
rel="stylesheet" type="text/css" />
    <script src="~/Static/common/js/jquery-1.4.2.min.js"
type="text/javascript"></script>
    <script src="~/Static/common/js/Core.js" type="text/javascript"></script>
    <script src="~/Static/common/js/jquery.autocomplete/jquery.autocomplete.min.js"
type="text/javascript"></script>
    <style type="text/css">
```



```

body
{
    font-size: 12px;
}

.formLabel
{
    float: left;
    width: 150px;
    text-align: right;
}
.formInput
{
    float: left;
}
</style>
</head>
<body>
<!-- Demo. 应用 AutoComplete 插件 -->
<div class="ui-widget ui-widget-content ui-corner-all" style="width: 700px; padding: 5px;">
    <h3>
        Demo. 应用 AutoComplete 插件
    </h3>
    <br style="clear: both" />
    <div class="formLabel">
        <label for="inputCityName">
            请输入城市拼音和汉字:</label>
        </div>
    <div class="formInput">
        <input id="inputCityName" name="inputCityName" type="text" />
    </div>
    <br style="clear: both" />
    <br style="clear: both" />
    <div class="formLabel">
        <label for="inputCityName">
            城市 ID:</label></div>
    <div class="formInput">
        <input id="inputCityId" name="inputCityId" type="text" /></div>
    <br style="clear: both" />
    <br style="clear: both" />
</div>
<script type="text/javascript">
    var thisPage = {
        initialize: function () { //加载时执行
            this.initializeDom();
            this.initializeEvent();

            $.getJSON("cityinfo.htm", null, this.initAutoComplete.bind(this))
        },
        initializeDom: function () { //初始化 DOM
            this.$inputCityName = $("#inputCityName");
            this.$inputCityId = $("#inputCityId");
        },
        initializeEvent: function () { //事件绑定
        },
        initAutoComplete: function (data) {
            var options = {
                minChars: 1,
                max: 500,
                width: 250,
                matchContains: true,

```

```
        formatItem: function (row, i, max) {  
            return i + "/" + max + ": " + row.CityNameEn + "\" [" +  
row.CityName + "];"  
        },  
        formatMatch: function (row, i, max) {  
            return row.CityNameEn + " " + row.CityName;  
        },  
        formatResult: function (row) {  
            return row.CityName;  
        }  
    };  
    this.$inputCityName.autocomplete(data, options); //对 inputCityName 应用 autocomplete 插件  
    this.$inputCityName.result(function (event, data, formatted) {  
        this.$inputCityId.val(data.ElongCityId);  
    }).bind(this);  
}  
  
$(thisPage.initialize());  
</script>  
</body>  
</html>
```

实现本实例的关键点就是在“initAutoComplete”方法中，对“\$inputCityName”对象应用了“autocomplete()”函数。autocomplete 插件的配置对象 option 是比较复杂的，接下来将详细地讲解 autocomplete 插件的使用。

### 11.4.2 准备数据源

首先要准备实现自动建议的数据源。本实例是通过发送 AJAX 请求获取 JSON 对象的。autocomplete()方法支持两个参数，第一个是 data，第二个是 options。

其中 data 参数可以是本实例中的一个数据变量，也可以是一个 URL。如果是 URL 则会每次都调用 AJAX 请求获取数据。

为了效率，更推荐在数据量允许的情况下，在页面加载后使用 AJAX 获取全部的数据，然后传递数据变量给 autocomplete 组件，如实例中所示。如果数据特别巨大无法在客户端加载，则只能每次都使用发送 AJAX 请求从服务器端获取部分数据。但是这会对服务器造成负担。

本实例将 JSON 格式的数据保存在了 CityInfo.htm 页面中。在 jQuery 的页面处理函数中，使用\$.getJSON()函数获取数据，在数据获取完毕后，使用获取到的数据初始化 autocomplete 控件。

### 11.4.3 设置关键函数

options 是可选项，因为本实例的数据源是一个多属性对象，所以必须设置下面几个关键的配置项后才能够使用，如表 11-1 所示。



表 11-1 autocomplete 控件关键配置项列表

配置项名称	配置项说明	配置项签名	签名参数说明
formatItem	对匹配的每一行数据使用此函数格式化, 返回值是显示给用户的数据内容	function(row, rowNum, rowCount, searchItem)	row: 当前行。the results row, rowNum: 当前行号, 从 1 开始 (注意不是索引, 索引从 0 开始) the position of the row in the list of results (starting at 1), rowCount: 总的行号 the number of items in the list of results searchItem: 查询使用的数据, 即 formatMatch()函数返回的数据格式的内容。我们在 formatMatch()函数中会设置程序内部搜索时使用的数据格式, 这个格式和给用户展示的数据是不同的
formatMatch	对每一行数据使用此函数格式化需要查询的数据格式。返回值是给内部搜索算法使用的。实例中用户看到的匹配结果是 formatItem 中设置的格式, 但是程序内部其实只搜索城市的英文和中文名称, 搜索数据在 formatMatch 中定义: <pre>return row.CityNameEn + " " + row.CityName;</pre>	function(row, rowNum, rowCount,)	同上
formatResult	此函数是用户选中后返回的数据格式。比如实例中只返回城市名给 input 控件: <pre>return row.CityName;</pre>	function(row, rowNum, rowCount,)	同上

#### 11.4.4 为控件添加 Result 事件函数

前面三个函数无法实现这类要求: 自动建议列表显示的是城市名称, 但是系统查询时使用城市 ID, 选中一个城市后需要将城市 ID 存储在一个隐藏域中。

所以 autocomplete 控件提供了 result()事件函数, 此事件会在用户选中某一项后触发:

```
this.$inputCityName.result(jQuery.proxy(function (event, data, formatted) {  
    this.$inputCityId.val(data.ElongCityId);  
}, this));
```

注意 jQuery.proxy()函数的使用。

result()函数签名为:

```
function(event, data, formatted)
```

参数列表如下:

Result 事件会为绑定的事件处理函数传递三个参数:

- ❑ event: 事件对象。event.type 为 result。
- ❑ data: 选中的数据行。
- ❑ formatted: 虽然官方的解释应该是 formatResult()函数返回的值, 但是实验结果是



formatMatch 返回的值。在本实例为: "Beijing 北京"。

#### 11.4.5 匹配中文

当前版本的 autocomplete 控件对中文搜索存在 bug, 原因是其搜索事件被绑定在 keydown 事件上, 当使用中文输入法输入“北”字时没有任何提示。本实例对原库做了修改, 将 keydown 事件修改为 keyup 事件, 即可完成对中文的智能提示搜索。另外需要将“matchContains”配置项设置为“true”, 因为我们的搜索格式是“Beijing 北京”, 默认只匹配开头的字符。

#### 11.4.6 其他注意事项

关于更多的配置项, 请参考官方文档: <http://docs.jquery.com/Plugins/Autocomplete/autocomplete#options>。

除了上面介绍的 autocomplete() 和 result() 函数, 还有如下函数:

- ❑ search(): 激活 search 事件。
- ❑ flushCache(): 清空缓存。
- ❑ setOptions( options ): 设置配置项。

### 11.5 小结

本章介绍 jQuery 中的插件, 首先讲解了如何开发插件。虽然 jQuery 的插件丰富多彩并且使用简单, 但是一定要慎重选择。对于专业的网站依然是建议自己开发插件。

在本章后面介绍了两个常用插件, 表单验证插件和自动建议插件。

有了对 jQuery 的全面了解, 读者可以以后自己尝试使用更多的 jQuery 插件或自行开发 jQuery 插件, 页面开发从此进入控件时代。





## 第 12 章



# 页面的华丽外衣——jQuery UI

使用 jQuery 类库，最常打交道的就是页面。jQuery 插件最常被使用的也是 UI 类插件。因为 jQuery 的插件开发和发布都很简单，导致插件的质量参差不齐，而且更新和维护也会随着开发者个人原因存在很多不确定性因素。

jQueryUI 是 jQuery 官方提供的 UI 类库，几乎包括了所有制作页面需要用到的功能，并且官方的开发质量、更新频率、向下兼容性都能够得到很好的保证。

通过本章的学习，你会发现，原来让页面变得相当专业是一件如此简单的事情，一个不懂美工的开发人员也可以做到！



## 12.1 jQuery UI 基础

首先,来了解一下 jQuery UI 及 jQuery UI 的分类,以便以后更加系统全面地了解 jQuery UI。

### 12.1.1 jQuery UI 简介

jQueryUI 是 jQuery 的官方插件,是在 jQuery 之上开发的专门用于 UI 交互的类库。使用 jQueryUI 最直接的用法就是使用它提供的 UI 控件,比如日历框、弹出框等。另外还可以基于 jQueryUI 提供的底层控件接口,比如可拖曳、可拉伸等来自自己开发控件。

jQueryUI 提供了丰富的“皮肤”(或叫做“主题”),所以可以快速地更换控件的样式。官方首页 <http://jqueryui.com/>。

下载地址 <http://jqueryui.com/download>。

示例和文档 <http://jqueryui.com/demos/>。

皮肤 <http://jqueryui.com/themeroller/>。

jQuery UI 的在线网站十分强大,在下载时可以组装自己想要的功能定制下载,如图 12-1 所示。

The screenshot displays the jQuery UI Themeroller interface. On the left, a list of components is shown, with a red box highlighting the 'UI Core' section. The 'UI Core' section includes four checked items: 'Core', 'Widget', 'Mouse', and 'Position'. Below this, the 'Interactions' section is visible, with five checked items: 'Draggable', 'Droppable', 'Resizable', 'Selectable', and 'Sortable'. On the right side, there are three panels: 'Theme' (labeled '选择皮肤') with a dropdown menu set to 'UI lightness', 'Version' (labeled '选择版本') with radio buttons for '1.8.2' (selected) and '1.7.3', and a 'Download' button (labeled '下载定制好的类库'). At the bottom right, there is a link to the 'Getting Started Guide'.

图 12-1 jQueryUI 定制下载页面

在说明文档中,提供的各种 Demo 示例都可以更换皮肤,如图 12-2 所示。



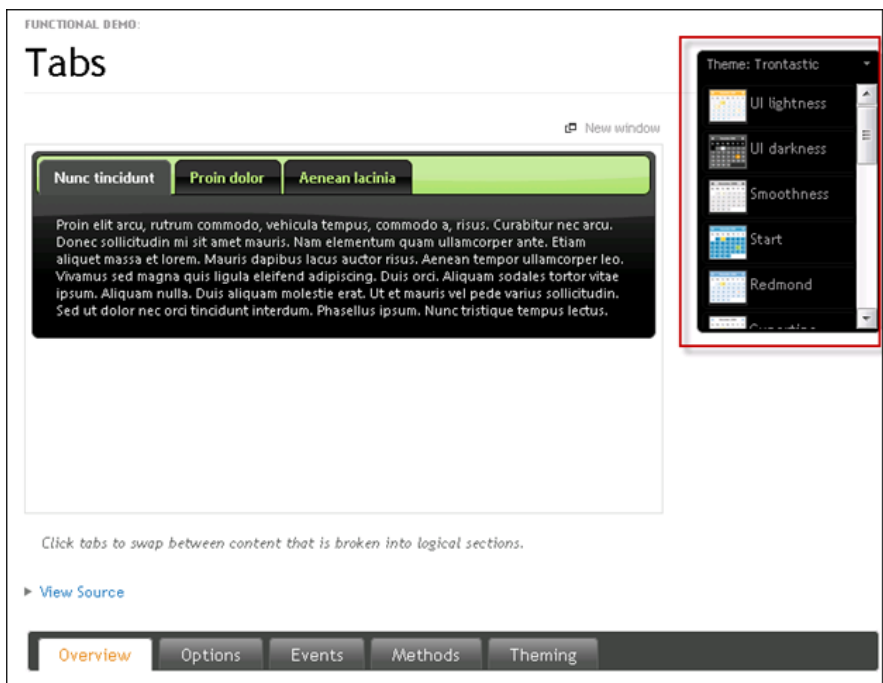


图 12-2 实例可更换皮肤

在最新的 jQuery UI 下载界面上已经没有了文件大小提示，可能是出于准确性不高的考虑。使用 jQuery UI 最大的问题就是其类库过大。一个包含了所有功能的最大 jQuery UI 类库脚本，有 200K，即使经过了 Gzip 压缩也依然有 50K 左右。正是因为如此，jQueryUI 类库才可以定制下载，你可以选择自己需要的控件和功能，这样能够有效地控制类库的大小。

### 12.1.2 jQuery UI 分类

jQuery UI 包括 4 个部分：

- ☐ Interactions（交互）。
- ☐ Widgets（页面控件）。
- ☐ Effects（页面效果）。
- ☐ Utilities（工具集）。

Interactions 表示提供底层的用户交互特效。比如可以为一个 div 添加拖动功能、重置大小功能。可以说是一组底层特效接口，用户可以在这些特效的基础上开发自己的控件。

Widgets 是在 Interactions 的基础上，由 jQuery UI 已经开发好的，可以快速使用的用户页面控件。比如弹出框控件、Tab 控件、日历控件等。

Effects 是指 easing 特效，在讲解 jQuery 动画时讲解了 jQuery 的 easing() 算法函数。jQuery UI 中就提供了各种不同的 easing() 函数及各种定义好的动画特效。

Utilities 提供了底层的工具函数，用来实现用户交互、各种控件及效果。

在本章将主要讲解 jQuery UI 提供的各种 Widgets 控件。在讲解各种 jQuery UI 控件的时候，将首先全面地讲解 datepicker 控件，包括列举其所有的属性、事件和方法。通过 datepicker 控件的讲解可以了解到如何学习一个控件。因为所有的控件都具有属性、事件和



方法。这些特性都是相通的。掌握了学习方法，就可以自己通过 jQueryUI 的官方文档进一步深入学习。对于其他控件的讲解将主要以实例为主。

## 12.2 Datepicker 日历控件

日历控件是每一个 Web 应用都不可或缺的控件。jQuery UI 提供了日历控件 Datepicker，如图 12-3 所示为日历控件的显示效果。

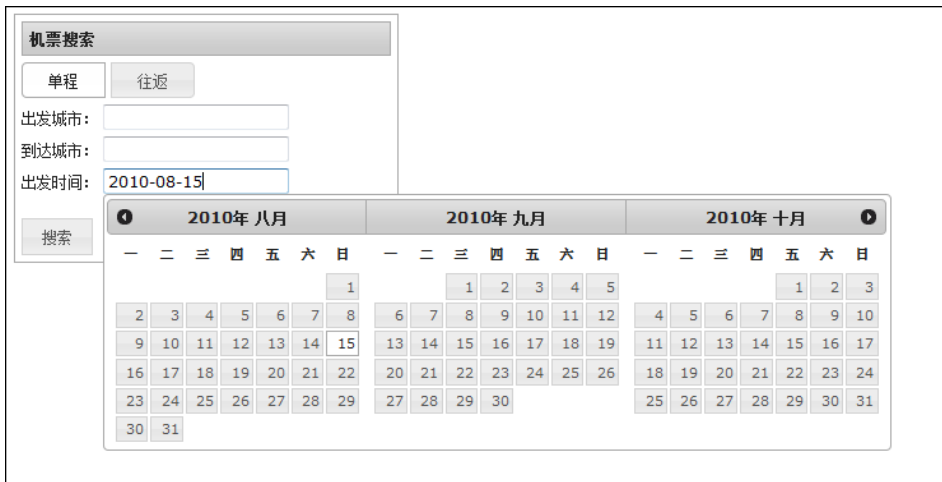


图 12-3 使用日历控件效果

Datepicker 日历框控件依赖 jQuery UI 的 UI Core 组件部分。Datepicker 控件的可配置性和扩展性非常高，比如可以配置日期格式、可以设置语言、可以设置日期范围等。并且暴露了各种事件和方法，便于加入自己的业务逻辑。Datepicker 还支持以下键盘操作。

- ❑ page up/down：更改月份。
- ❑ ctrl+page up/down：更改年份。
- ❑ ctrl+home：回到本月。
- ❑ ctrl+left/right：选择上一天或下一天。
- ❑ ctrl+up/down：选择上一周或下一周。
- ❑ enter：选中日期。
- ❑ ctrl+end：关闭日历并清空选中日期。
- ❑ esc：关闭日历框不选择日期。

下面将首先通过实例来了解 Datepicker 的基本用法，在懂得如何上手后，再来全面地学习 Datepicker 控件的各个属性和方法。有关日历框控件最新的官方文档，可以在下列地址查看：

<http://jqueryui.com/demos/datepicker/>。

### 12.2.1 应用实例

要使用 Datepicker，需要引用 jQuery UI 的皮肤、jQuery 类库、jQuery UI 类库（至少包



含 jQuery UI Core 和 Datepicker 组件部分) 三个部分, 这些在 jQuery UI 的下载包中都会包括。在 head 中加入:

```
<link href="../Static/common/css/Smoothness/style.css" rel="stylesheet" type="text/css" />
<script src="../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
<script src="../Static/common/js/jquery-ui-1.8.2.custom.min.js" type="text/javascript"></script>
```

假设页面上存在如下 HTML 区域:

```
<div>
  <label for="datepickerStart">出发日期: </label><input id="datepickerStart" />&nbsp;  
  <label>隐藏值: </label><input id="alternate" />
</div>
<div>
  <label for="datepickerEnd">结束日期: </label><input id="datepickerEnd" />
</div>
<div id="divDatepicker"></div>
```

下面介绍 Datepicker 控件不同的用法。

首先通过 jQuery 选择器, 获取页面元素:

```
var $dateStart = jQuery("#datepickerStart");
var $dateEnd = jQuery("#datepickerEnd");
```

最简单的就是使用默认设置, 弹出一个标准的日历控件:

```
$dateStart.datepicker(); //默认样式
```

效果如图 12-4 所示。



图 12-4 日历框默认样式

默认的样式有以下几个特点。

- ❑ 使用了引入的 jQuery 皮肤样式, 本例中使用了 “Smoothness” 灰色皮肤。
- ❑ 日期格式为 “mm/dd/yyyy”。
- ❑ 默认是英文。
- ❑ 只显示一个日历。
- ❑ 可以通过箭头更换月份。

同所有的 jQuery UI 控件一样, Datepicker 提供了 options 属性用来设置日历框 (比如设置日历框可选择的最小日期)、提供了 Events 用来在发生各种行为时编写逻辑 (比如在选择日期时弹出提示框)、提供方法用于操作日历框 (比如隐藏日历框)。

通过传递不同的 options 属性, 可以为日历框设置不同的样式和行为。

首先最重要的就是语言和格式。在下载 jQuery UI 类库中，可以找到类似：

jquery.ui.datepicker-fr.js

这样的语言文件。其中 fr 就是指语言类型。找到中文语言包，将其中的下列内容复制到页面上：

```
jQuery(function ($) {
    if ($.datepicker) {
        $.datepicker.regional['zh-CN'] = {
            closeText: '关闭',
            prevText: '<#x3c;上月',
            nextText: '下月>#x3e;',
            currentText: '今天',
            monthNames: ['一月', '二月', '三月', '四月', '五月', '六月',
                '七月', '八月', '九月', '十月', '十一月', '十二月'],
            monthNamesShort: ['一', '二', '三', '四', '五', '六',
                '七', '八', '九', '十', '十一', '十二'],
            dayNames: ['星期日', '星期一', '星期二', '星期三', '星期四', '星期五', '星期六'],
            dayNamesShort: ['周日', '周一', '周二', '周三', '周四', '周五', '周六'],
            dayNamesMin: ['日', '一', '二', '三', '四', '五', '六'],
            weekHeader: '周',
            dateFormat: 'yy-mm-dd',
            firstDay: 1,
            isRTL: false,
            showMonthAfterYear: true,
            yearSuffix: '年'
        };
        $.datepicker.setDefaults($.datepicker.regional['zh-CN']);
    }
});
```

注意这段脚本一定要放在 jQuery UI 库之后。因为这实际上是作用在 Datepicker 控件的一个全局变量 \$.datepicker 上的。通过设置全局属性，将页面上所有的日历框变成了中文，如图 12-5 所示。



图 12-5 切换到中文语言的日历控件

除了日历框的文字变化之外，选中日期后的格式也变成了 “yyyy-mm-dd”。

日历框控件可以在选择日期后，将日期以不同的格式填充到另外一个 input 元素上：

```
$dateStart.datepicker({ altField: '#alternate', altFormat: 'DD, d MM, yy' }); //选中日期后，用不同日期格式填充另外一个 input
```



其中 altField 是要填充元素的选择器。altFormat 是填充的日历格式。此示例的效果如图 12-6 所示。



图 12-6 日历控件填充隐藏域

日历框控件的另外两个重要功能，一个是限制选择的日期，另外可以同时多排显示多个月份：

```
$dateStart.datepicker({ numberOfMonths: 3, minDate: +0, maxDate: '+2M +1D' }); //限制可选择的日期
```

通过设置日历框的 minDate 和 maxDate 属性，可以设置日历框选择的起始和结束日期。这两个属性都支持三种类型的对象，分别是 Date 日期类型、数字和字符串。

JavaScript 中原生的 Date 日期类型其本身就是一个事件对象，唯一要注意的是 Date 对象中的月份是 0~11，也就是说，如果想设置 2010 年 2 月 1 日，则需要创建对象：

```
new Date("2010", "1", "1");
```

如果传递数字，表示增加或减少的天数，正数是增加，比如“+7”，负数是减少，比如“-5”。

如果传递字符串，则可以分别加减年、月、日。比如“+1y +2m -3d”。其中 y 表示年，m 表示月，d 表示天。

“numberOfMonths”属性用来控制一次并排显示多少个月份。

上面的例子中，显示了 3 个月份，可选日期范围是今天到以后的 2 个月零 1 天，效果如图 12-7 所示。



图 12-7 多日历框及限制可选日期

本示例的完整代码参见：

代码 12-1 日历控件示例【jQueryStorm.Web/chapter12/ Demo-UI-Datepicker.htm】



下面是日历框控件的另外几种常用用法：

```
$dateStart.datepicker({ dateFormat: "yy-mm-dd" }); //修改返回的日期格式为"yy-mm-dd"
$dateStart.datepicker({ showButtonPanel: true }); //显示按钮
$dateStart.datepicker({ changeMonth: true, changeYear: true }); //显示更换月份和年份
//在本月中，显示上一个月和下个月的部分日期
$dateStart.datepicker({ showOtherMonths: true, selectOtherMonths: true });
$dateStart.datepicker({ showWeek: true, firstDay: 1 }); //显示每一周是今年的第几周
$dateStart.datepicker({ showAnim: "slideDown" }); //使用不同的动画效果显示日历
```

由于日历框的功能十分强大，无法在这里列举每一种应用。下面列出日历框相关的参数、事件和方法列表。

### 12.2.2 日历框参数

所有的 jQuery UI 控件，都使用 options 对象设置参数。

在初始化 Datepicker 控件时，可以传递一个包含了若干属性的对象，用来设置 Datepicker 的各种属性，比如：

```
$dateStart.datepicker({ dateFormat: "yy-mm-dd" }); //修改返回的日期格式为"yy-mm-dd"
```

上面的例子中，传递的 options 对象只有一个属性 “dateFormat”，则日历框控件的日期格式已经被修改。

设置 options，首先可以在控件初始化时，传递一个 map 对象（含有多个属性的对象）实现：

```
$dateStart.datepicker({
    altField: "#alternate",
    altFormat: "DD,d MM, yy"
});
```

也可以单独地修改某一个 options 属性值：

```
$.dateStart.datepicker("option", "altField", "#alternate");
```

可以获取某一个 options 属性值：

```
$.dateStart.datepicker("option", "altField");
```

通过控制控件的 options 属性，可以控制控件的显示和行为。目前使用 options 对象保存参数，不仅仅应用在 jQuery UI 的所有控件上，几乎所有的 jQuery 控件也都是这样使用的。

对于每一个 options 属性的作用，可以参考每一个控件的帮助文档。Datepicker 控件将列举出它的所有属性，对于其他控件的 options 属性请自行参阅官方文档。

日历控件的 options 对象的属性说明，如表 12-1 所示。

表 12-1 Datepicker 日历控件 options 参数说明

参数名称	数据类型	默认值	说 明
disabled	Boolean	False	通过 disabled 属性可以控制日历框是否有效
altField	String	“ ”	另外一个元素的选择器表达式，在选中日期时使用 altFormat 属性定义的日期格式将选中日期放入此元素
altFormat	String	“ ”	填充 altField 表达式选中元素的日期格式





(续表)

参数名称	数据类型	默认值	说 明
appendText	String	“ ”	在日期控件的页面元素后面加入一段文字。经测试是加入了一段 span，比如： \$( ".selector" ).datepicker({ appendText: '(yyyy-mm-dd)' }); 加入的 HTML 为： <span class="ui-datepicker-append">(yyyy-mm-dd)</span>
autoSize	Boolean	false	是否自动设置 input 元素的大小，如果设置为 true 将根据 dateFormat 属性格式化后的日期字符串重置 input 元素的大小
buttonImage	String	“ ”	弹出按钮图片的 URL 地址。如果设置了此属性，则 buttonText 属性会此图片的 alt 和 title 属性
buttonImageOnly	Boolean	false	默认情况下，如果将 “showOn” 属性设置为了 “button” 或 “both”，将在 input 后添加一个 button 元素，也可以用来显示日历。默认情况下添加 buttonImage 会在 button 内部添加一张图片，如果将此属性设置为 true 则会将 button 元素变成 img 元素
buttonText	String	“...”	在 button 上启用的文字
calculateWeek	Function	\$.datepicker.iso8601Week	根据日期计算本周是第几周的函数。默认是根据 ISO 8601 标准实现的：每周一是一周的开始，每年的第一是包含本年第一个星期四的一周
changeMonth	Boolean	false	true 表示允许从下拉框更改月份
changeYear	Boolean	false	true 表示允许从下拉框更改年份
closeText	String	“Done”	关闭按钮的文字，这是一个区域化属性，比如如果选择了中文语言包，则会使用此语言包的此属性。需要使用 “showButtonPanel” 属性显示关闭按钮时此属性才起作用
constrainInput	Boolean	true	如果设置为 true，则 input 只能输入 dateFormat 定义的格式中允许的字符（注意只是限制字符，不能限制格式）
currentText	String	“Today”	“今天” 按钮显示的文字，与 “closeText” 属性一样是区域化的文字之一，需要配合 “showButtonPanel” 属性使用
dateFormat	String	“mm/dd/yy”	日期格式字符串，控制选中日期后填充的格式。区域化属性之一。比如中文语言包会自动改成 “yy-mm-dd”，可以从下面的地址中查看所有支持的格式： <a href="http://docs.jquery.com/UI/Datepicker/formatDate">http://docs.jquery.com/UI/Datepicker/formatDate</a>
dayNames	Array	['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']	每天是星期几的全文字。区域属性之一。从星期日开始。此文字是星期的全名，会出现在选中后的 input 文字中，以及当鼠标停留在日历框上的 dayNamesMin（星期缩写）上时显示的文字。使用中文语言包会设置为： ['星期日', '星期一', '星期二', '星期三', '星期四', '星期五', '星期六']
dayNamesMin	Array	['Su', 'Mo', 'Tu', 'We', 'Th', 'Fr', 'Sa']	星期最简单缩写。在弹出框上会显示此文字。区域属性之一。中文语言包会设置为： ['日', '一', '二', '三', '四', '五', '六']





(续表)

参数名称	数据类型	默认值	说 明
dayNamesShort	Array	['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']	星期缩写。和 dayNamesMin 的不同是它只用在选中日期后填充 input 时使用。中文语言包会设置为: ['周日', '周一', '周二', '周三', '周四', '周五', '周六']
defaultDate	Date, Number, String	null	如果 input 没有日期, 则日历框打开后会高亮显示 defaultDate 中设置的日期。此高亮是鼠标悬浮在某一个日期上时的高亮效果, 而不是选中日期的高亮效果。如果直接按下回车, 则会在 input 中填充此日期
duration	String, Number	"normal"	日历框动画效果的渐变时间
firstDay	Number	0	设置每周第一天是星期几。0 表示星期日。是区域相关属性。中文语言包中将此值设置为 1
gotoCurrent	Boolean	false	如果设置为 true, “回到当前日期” 连接将改为回到选中的日期, 默认为回到今天
hideIfNoPrevNext	Boolean	false	如果设置为 true, 则当超过了 minDate 和 maxDate 设置的日期时, 上一个月和下个月的链接将不显示。(注意仅仅是控制是否显示按钮, 如果月份小于 minDate 设置的月份, 即使单击“上一月”按钮也无效)
isRTL	Boolean	false	是否是 “Right To Left” 右到左。有的语言是从右到左阅读的。这是一个区域属性。中文语言包此属性为 false
maxDate	Date, Number, String	null	可以选择的最大日期
minDate	Date, Number, String	null	可以选择的最小日期
monthNames	Array	['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December']	每个月的全名。区域属性。中文语言包为: ['一月', '二月', '三月', '四月', '五月', '六月', '七月', '八月', '九月', '十月', '十一月', '十二月']
monthNamesShort	Array	['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']	每个月的缩写。区域属性。中文语言包为: ['一', '二', '三', '四', '五', '六', '七', '八', '九', '十', '十一', '十二']



(续表)

参数名称	数据类型	默认值	说 明
navigationAsDate Format	Boolean	false	如果设置为 true, 则 formatDate()函数会作用在 nextText、prevText 和 currentText 三个属性上, 允许它们使用日期来作为文字内容。比如把“一个月”按钮的文字, 改成显示“下月: 八月”这样的格式, 可以设置: navigationAsDateFormat:true nextText : "下月: MM"
nextText	String	“Next”	下一个月按钮的提示文字
numberOfMonths	Number, Array	1	一次显示几个月份。如果传递数字则表示在一行中显示几个月份。也可以传递一个含有两个元素的数组。比如: [2,3]表示显示两行, 每行显示 3 个月份
prevText	String	“Prev”	上一个月按钮的提示文字
selectOtherMonths	Boolean	false	只有当 showOtherMonths 设置为 true 时生效, 在当月显示的其他月份的日期将可以被选中
shortYearCutoff	String, Number	“+10”	设置截止年份的值。如果是 (0~99) 的数字则以当前年份开始算起; 如果为字符串, 则相应地转为数字后再与当前年份相加。当超过截止年份时, 则被认为是上一个世纪
showAnim	String	“show”	设置显示、隐藏日期插件的动画的名称
showCurrentAtPos	Number	0	设置当多月份显示的情况下, 当前月份显示的位置。自顶部/左边开始第 x 位
showMonthAfterYear	Boolean	false	是否在面板的头部年份后面显示月份
showOn	String	“focus”	设置什么事件触发显示日期插件的面板, 可选值有 focus、button 和 both
showOptions	Options	{}	如果使用 showAnim 显示动画效果, 可以通过此参数来增加一些附加的参数设置
showOtherMonths	Boolean	false	是否在当前面板显示上、下两个月的一些日期数
stepMonths	Number	1	当单击上/下一月时, 一次翻几个月
yearRange	String	“-10:+10”	控制年份的下拉列表中显示的年份数量, 可以是相对当前年 (-nn:+nn), 也可以是绝对值 (-nnnn:+nnnn)

### 12.2.3 日历框事件

jQuery UI 控件的事件, 也都是包含在 options 中的属性。比如设置 “beforeShow” 事件:

```
$dateStart.datepicker({ beforeShow: function (input, inst) {
    input.value = "1984-10-10";
}
});
```

设置事件和设置控件的所有 options 属性一样。事件都是函数, 在控件发生特定行为时触发。比如 DatePicker 日历框控件的 “beforeShow” 事件会在日历框显示之前触发。上面的例子通过此事件来初始化日历框的日期。

因为事件也是 options 对象的一个属性, 所以也可以同其他 options 属性一样获取到:

```
$dateStart.datepicker("option", "beforeShow")
```



但是不能使用 set 访问器设置。要设置事件还是应该在 Datepicker 控件初始化时执行。比如本节一开始的例子。

下面是 Datepicker 日历控件的事件列表，如表 12-2 所示。对于其他的控件，不会再给出这样的列表。每个 jQuery UI 控件的 options 属性、事件和函数都可以在此控件的说明文档中找到。学会了解决问题的方法才是最重要的。

表 12-2 Datepicker 日历框控件事件列表

事件名称	事件签名	参数说明	事件说明
beforeShow	function( input, inst)	input: input 元素 inst: datepicker 对象 返回值: options 对象	在日期控件显示面板之前，触发此事件，并返回当前触发事件的控件实例对象
beforeShowDay	function( date)	date: 日期 返回值: Array 对象	在日期控件显示面板之前，每个面板上的日期绑定时都触发此事件，参数为触发事件的日期。调用函数后，必须返回一个数组：[0]此日期是否可选(true/false)，[1]此日期的 CSS 样式名称（传递""表示使用默认），[2]当鼠标移至上面出现一段提示的内容
onChangeMonthYear	function( year, month, inst)	year: 选中的年 month: 选中的月 inst: datepicker 对象	当年份或月份改变时触发此事件，参数为改变后的年份月份和当前日期插件的实例。this 表示 input 元素
onClose	function( dateText, inst)	dateText: 选中的日期 inst: datepicker 对象	当日期面板关闭后触发此事件（无论是否有选择日期），参数为选择的日期和当前日期插件的实例。dateText 为选中的日期，如果没有选择日期则为“”，this 表示 input 元素
onSelect	function( dateText, inst)	dateText: 选中的日期 inst: datepicker 对象	当在日期面板选中一个日期后触发此事件，参数为选择的日期和当前日期插件的实例。this 表示 input 元素

## 12.2.4 日历框方法

事件是控件内部调用的函数，而控件方法则是供客户端调用的功能函数。

比如常用的用来设置 options 属性的语句其实就是使用了一个方法：

```
.datepicker( "option", optionName , [value] )
```

用来获取 options 属性的语句同样是调用了一个方法：

```
.datepicker( "option", options )
```

函数可以用来在客户端操作控件对象。注意这些函数都是控件对象的实例方法，只能在控件对象上调用。比如可以手工地控制一个 Datepicker 对象显示或隐藏：

```
$dateStart.datepicker("show");  
$dateStart.datepicker("hide");
```

如表 12-3 所示为 Datepicker 日历框控件的方法列表。

表 12-3 Datepicker 日历框控件方法列表

方法名称	用 法	说 明
destory	.datepicker( 'destroy' )	完全地移除 Datepicker 对象，这个函数返回元素初始化之前的状态



(续表)

方法名称	用 法	说 明
disable	<code>.datepicker( 'disable' )</code>	让一个日历框失效
enable	<code>.datepicker( "enable" )</code>	让一个日历框有效
option	<code>.datepicker( "option", optionName , [value] )</code>	设置某一个 options 属性
option	<code>.datepicker( "option", options )</code>	获取某一个 options 属性
widget	<code>.datepicker( "widget" )</code>	返回引用了 “.ui-datepicker” 样式的元素
dialog	<code>.datepicker( "dialog", date , [onSelect] , [settings] , [pos] )</code>	在一个 dialog 对话框控件中打开 Datepicker 日历框控件。 dateText: 初始化的日期。Date 对象会字符串。 onSelect: 当日期被选中后的回调函数。这个函数接收日期文本和日历框实例作为参数。 settings: 新的用来设置 Datepicker 的参数集合。 pos: dialog 控件的 top/left 属性, 或者是一个鼠标事件对象包含位置坐标。如果没有传递则 dialog 控件将显示在中央
isDisabled	<code>.datepicker( "isDisabled" )</code>	判断日历控件是否有效
hide	<code>.datepicker( "hide" )</code>	隐藏日历框控件
show	<code>.datepicker( "show" )</code>	显示日历框控件
refresh	<code>.datepicker( "refresh" )</code>	重新绘制日历, 如果外部有了某些更改则需要调用 refresh() 方法
getDate	<code>.datepicker( "getDate" )</code>	返回 Datepicker 当前的日期, 如果没有日期选中则返回 null
setDate	<code>.datepicker( "setDate", date )</code>	设置 Datepicker 当前的日期。新的日期可以是 Date 对象、符合当前日期格式的字符串(比如“1984-10-10”)、一个数字表示在今天的基础上增加几天(比如+7)、一个表示时间的字符串(y 表示年, m 表示月, w 表示星期, d 表示天。比如: “+1m +7d”)或者传递 null 值表示清空选中的日期

现在已经完全地列举了日历框控件的所有属性、事件和方法。读者是不是感觉太过复杂呢? 灵活和简单往往存在着冲突。Datepicker 在设计时, 就着重考虑了灵活性, 因为设计者无法知道使用者的使用场景。通过提供 Datepicker 的默认值, 让使用者尽可能简单地使用。只有当使用者发现各种默认值无法满足自己的需求时, 才去翻阅文档考虑应该修改哪些属性或事件。这样就能在灵活和易用之间找到一种平衡。

当然也可以根据自己项目的使用场景, 开发自己的控件。在第 13 章中将介绍如何基于 jQuery 开发自己的脚本控件。

## 12.3 Dialog 对话框控件

### 12.3.1 对话框应用场景

对话框是最常用、最实用的功能。先来看一下艺龙网上的一些应用场景。

关于艺龙: 艺龙是在线旅游行业领先服务商, 提供酒店、机票和度假产品的预订。也是本书作者曾经所在的公司。网址是 [www.elong.com](http://www.elong.com), 当您访问网站时不一定能找到本书截图的特效。因为艺龙一直在为了用户体验做持续的改进。



- (1) 静态提示类对话框，对话框的内容是固定的，如图 12-8 所示。  
(2) 动态提示类对话框，对话框内容是根据事件源变化的，如图 12-9 所示。



图 12-8 静态内容对话框



图 12-9 动态内容对话框

- (3) 遮罩类对话框，对话框弹出时背景变灰并不可选，如图 12-10 所示。



图 12-10 遮罩类对话框

### 12.3.2 应用实例

使用 jQuery UI 的 Dialog 组件可以轻松实现上面三种效果。

Dialog 组件的主要特点是可以拖动 (Draggable)，可以改变大小 (Resizable)。

Dialog 对话框的使用也十分简单，选中了一个元素后，可以通过对这个元素使用“.dialog()”让其变成一个对话框，通过传递各种 options 属性来修改对话框的各种行为。

通常一个对话框是一个 div 元素：

```
<div id="divTip" title="自定义标题">
  <p>弹出层</p>
</div>
```

下面的语句将使用默认的 options 属性生成一个对话框。



当然这只是最简单的应用。下面通过一个完整实例来快速上手 dialog 对话框组件。示例完整代码可以在下列路径中找到：

另外一种常见的遮罩类弹出层，即弹出层显示后，页面上除了弹出层以外的元素都不能操作。

弹出层的 HTML 代码如下:



```
frameborder="0"></iframe>
</div>
```

弹出层就是一个个 div 元素。会根据需要显示或隐藏。

准备好了 HTML 元素, 接下来就是应用 jQuery UI 的 Dialog 控件。首先, 在 initializeDom 中, 获取稍后需要操作的页面元素。

```
initializeDom: function () { //初始化 DOM
    this.$spanShowTip = $("span[id^=spanShowTip]");
    this.$spanShowDataTip = $("span[id^=spanShowDataTip]");
    this.$btnShowIframe = $("#btnShowIframe");
    this.$divTip = $("#divTip");
    this.$divIframe = $("#divIframe");
}
```

接下来, 也是其中最重要的一步, 就是对两个弹出层对象应用 Dialog() 函数:

```
//初始化提示类弹出层
this.$divTip.dialog({
    show: 'slide',
    bgiframe: false,
    autoOpen: false
});
//初始化遮罩类弹出层
this.$divIframe.dialog({
    show: null,
    bgiframe: false,
    autoOpen: false,
    draggable: true,
    resizable: false,
    modal: true,
    width: 500,
    height: 300
});
```

有关 Dialog() 函数的使用, 将在后面详细讲解。

然后在 initializeEvent 事件中为页面上的元素添加事件绑定, 代码如下:

```
initializeEvent: function () { //事件绑定
    //静态提示类弹出层
    this.$spanShowTip.click(function (event) {
        $("*").stop(false, true);
        event.stopPropagation();
        this.$divTip.dialog("close");

        var scrollTop = $(document).scrollTop();
        var scrollLeft = $(document).scrollLeft();
        var top = $(event.target).offset().top + $(event.target).height() - scrollTop + 6;
        var left = $(event.target).offset().left - scrollLeft;

        this.$divTip.dialog("option", "position", [left, top]);
        this.$divTip.dialog("open");
    }).proxy(this);

    //动态提出类弹出层
    this.$spanShowDataTip.click(function (event) {
        $("*").stop(false, true);
        this.$divTip.dialog("close");
        event.stopPropagation();

        var scrollTop = $(document).scrollTop();
```



```

var scrollLeft = $(document).scrollLeft();
var top = $(event.target).offset().top + $(event.target).height() - scrollTop + 6;
var left = $(event.target).offset().left - scrollLeft;

this.$divTip.html($(event.target).attr("data"));
this.$divTip.dialog("option", "position", [left, top]);
this.$divTip.dialog("open");
}.proxy(this));

//遮罩类弹出层
this.$btnShowIframe.click(function (event) {
    event.preventDefault();
    event.stopPropagation();
    this.$divIframe.dialog("open");
}.proxy(this));

//单击自身取消冒泡
this.$divTip.bind("click", function (event) {
    event.stopPropagation();
});
this.$divIframe.bind("click", function (event) {
    event.stopPropagation();
});

//document 对象单击隐藏所有弹出层
$(document).bind("click", function (event) {
    this.$divTip.dialog("close");
    this.$divIframe.dialog("close");
}.proxy(this));
}

```

静态对话框效果如图 12-11 所示。

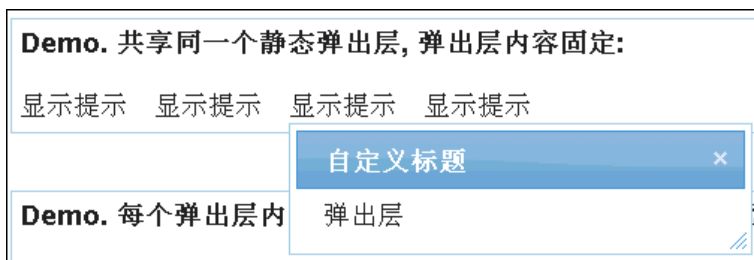


图 12-11 Dialog 静态对话框

动态提示类对话框效果如图 12-12 所示。

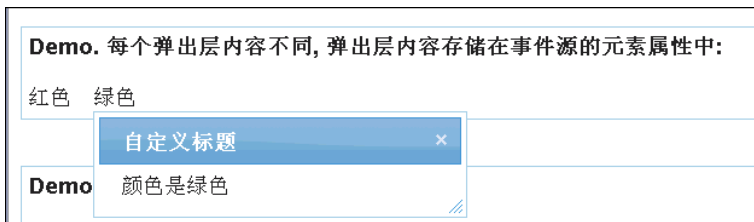


图 12-12 Dialog 动态对话框

遮罩类对话框效果如图 12-13 所示。







图 12-13 Dialog 遮罩类对话框

接下来对此示例中的关键点进行讲解。

### 12.3.3 计算对话框位置

弹出式对话框最重要的是计算弹出框的位置。通过事件对象的 `event.target` 属性可获取到事件源，使用 `offset()` 函数计算事件源相对于 `document` 的位置，使用 `scrollTop` 和 `scrollLeft` 计算鼠标滚动条滚过的距离，从而计算出对话框的位置：

```
var scrollTop = $(document).scrollTop();
var scrollLeft = $(document).scrollLeft();
var top = $(event.target).offset().top + $(event.target).height() - scrollTop + 6;
var left = $(event.target).offset().left - scrollLeft;
```

因为是相对于 `document`，即页面左上角的位置，所以需要将对话框放在 `Body` 元素中的第一层，即父类就是 `body`。如果包含在其他元素中，需要确定任何一个父类的 `position` 样式没有设置为 `relative`。

### 12.3.4 取消冒泡和浏览器默认行为

如果为 `document` 对象绑定了单击后关闭对话框的事件，那么就一定要取消事件的冒泡。使用 `event` 对象的 `stopPropagation()` 方法可以取消冒泡。

```
event.stopPropagation();
```

对于具有默认行为的元素，比如提交按钮的提交表单行为，`<a>` 元素的超链接行为等，如果在这些元素上应用事件，还需要取消它们的默认行为：

```
event.preventDefault();
```

### 12.3.5 设置动画效果与取消动画

通过设置 `dialog` 的配置项的 `show` 属性，可以设置显示 `Dialog` 时候的动画效果：

```
$('.selector').dialog({ show: 'slide' });
```

`show` 默认为 `null` 即无动画，可以是使用下列值：

```
'blind', 'clip', 'drop', 'explode', 'fold', 'puff', 'slide', 'scale', 'size', 'pulsate'.
```





对于这些动画的效果， 可以在此页观看：<http://jqueryui.com/demos/show/>。

当一个动画效果执行时，如果用户在此对这个元素进行操作，就会出现各种问题，比如定位不准确等。所以如果应用了动画，在对其操作时需要使用 `stop()` 函数来停止动画，通常是停止所有元素的动画：

```
$("#*").stop(false, true);
```

但是即使停止了动画再进行操作，如果操作太快也容易产生问题。所以至于是否使用动画需要经过权衡决定。

### 12.3.6 动态提示类对话框的数据传递

动态提示类对话框的数据是不同的，本文实例使用的是将数据存储在元素属性 `data` 上：

```
<span id="spanShowDataTip1" data="颜色是红色">红色</span>
```

这是一种简单直观的方式。比较容易编程实现（尤其是在使用 MVC 编程模型的时候）。另外可以使用 `jQuery.data` 为控件添加数据。

### 12.3.7 更换主题

注意实例中的对话框没有为其编辑任何样式，但是显示出来后已经被美化过了。这是因为引用了 jQuery UI 的主题。jQuery UI 包括下列主题：

```
<!--black-tie, blitzer, blitzer, dot-luv, excite-bike, hot-sneaks, humanity, mint-choc, redmond, smoothness, south-street, start, swanky-purse, trontastic, ui-darkness, ui-lightness, vader-->
```

可以在此地址查看各个主题的效果：<http://jqueryui.com/themeroller/#themeGallery>。

## 12.4 Tab 标签控件

Tab 标签控件可以不刷新页面即可实现在页面中的不同标签间切换，效果如图 12-14 所示。



图 12-14 Tab 控件效果

本实例通过 jQuery UI 的 Tabs 组件实现。Tabs 组件的使用与 Dialog 一样简单， 默认的配置即可实现最简单的功能， 通过设置更多的 options 可以实现更复杂的应用。

### 12.4.1 应用实例

Tab 标签控件的使用最关键的是要准备好合乎要求的 HTML 元素：

```
<div id="tabs2" style="width: 300px;">
  <ul>
    <li><a href="#tabs2-1">One</a></li>
    <li><a href="#tabs2-2">Two</a></li>
    <li><a href="#tabs2-3">Three</a></li>
  </ul>
```



```
<div id="tabs2-1">
  <p>Tab1 内容</p>
</div>
<div id="tabs2-2">
  <p>Tab2 内容</p>
</div>
<div id="tabs2-3">
  <p>Tab3 内容</p>
</div>
</div>
```

下面使用默认设置，上面的元素变成 Tab 标签：

```
$("#tabs1").tabs();
```

有关 HTML 元素的格式要求，随后将详细讲解。

首先来看一个完整的示例，示例代码可以在下列路径中找到：

【代码路径：jQueryStorm.Web/chapter12/ Demo-UI-Tab.htm】

本实例演示了三个 Tab 控件的使用：

第一个 Tab 控件的例子使用了默认的行为，在此 Tab 的第二个标签里实现了使用 AJAX 获取数据。HTML 代码片段如下：

```
<!--Demo.默认 Tab 与 Ajax Tab -->
<div id="tabs1" style="width:300px;">
  <ul>
    <li><a href="#tabs1-1">One</a></li>
    <!-- Ajax Tab -->
    <li><a href="TabData.htm">Two</a></li>
    <li><a href="#tabs1-3">Three</a></li>
  </ul>
  <div id="tabs1-1">
    <p>Tab1 内容</p>
  </div>
  <div id="tabs1-3">
    <p>Tab3 内容</p>
  </div>
</div>
```

脚本片段如下：

```
//默认 Tabs
$("#tabs1").tabs();
```

效果如图 12-15 所示。

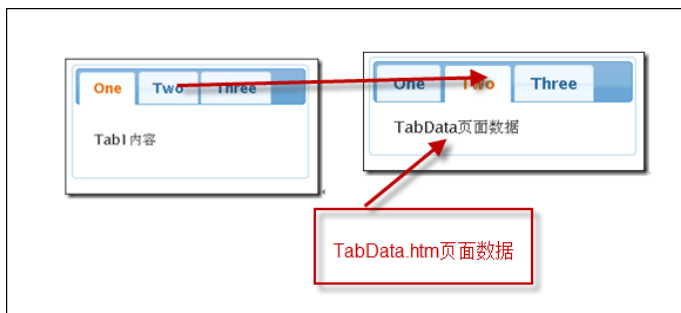


图 12-15 实现 AJAX 功能的 Tab 控件

第二个 Tab 控件，实现了收放功能，单击一个 tab 切换到此 Tab 页，再次单击则进行 Tab 页收缩/张开状态的切换。

HTML 片段如下：

```
<!--Demo. 可折叠的 Tab -->
<div id="tabs2" style="width: 300px;">
  <ul>
    <li><a href="#tabs2-1">One</a></li>
    <li><a href="#tabs2-2">Two</a></li>
    <li><a href="#tabs2-3">Three</a></li>
  </ul>
  <div id="tabs2-1">
    <p>Tab1 内容</p>
  </div>
  <div id="tabs2-2">
    <p>Tab2 内容</p>
  </div>
  <div id="tabs2-3">
    <p>Tab3 内容</p>
  </div>
</div>
```

在编写代码时只需要多传递一个参数即可：

```
//可折叠的 Tabs
$("#tabs2").tabs({
  collapsible: true
});
```

收缩后的 Tab 页效果如图 12-16 所示。

第三个 Tab 控件，可以在鼠标滑动即切换的 Tab 标签。控件的 HTML 元素部分和强两个基本相同。只需要多传递一个 event 参数即可。脚本片段如下：

```
//鼠标滑动即切换的 Tabs
$("#tabs3").tabs({
  event: "mouseover"
});
```

效果如图 12-17 所示。



图 12-16 可收放的 Tab 控件

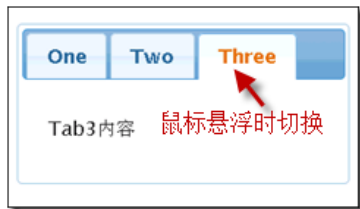


图 12-17 鼠标滑动时切换 Tab 标签

接下来将详细地讲解 Tab 控件使用时的注意事项。

## 12.4.2 注意 HTML 结构

使用 Tab 控件，尤其是默认的 options 设置时，一定要注意 HTML 的格式。只有格式正确才能被正确地转换为 tab 控件。其中最外层是一个 div 元素，里面嵌套一个 ul 元素，ul



作为 tab 页的菜单。ul 中的每一个 li 元素将成为页面上的一个 tab 菜单项。li 元素中通过添加 a 元素，并指定 href 属性与接下来的 div 关联。比如：

```
<li><a href="#tabs2-1">One</a></li>
```

此 li 将作为菜单项，此菜单中的内容是 div 内 ID 为“tabs2-1”的 div 中的内容。href 属性也可以是一个页面，如果是页面地址则将启用 AJAX 模式：

```
<li><a href="TabData.htm">Two</a></li>
```

下面看一个完整的 HTML 结构：

```
<div id="tabs2" style="width: 300px;">
  <ul>
    <li><a href="#tabs2-1">One</a></li>
    <li><a href="#tabs2-2">Two</a></li>
    <li><a href="#tabs2-3">Three</a></li>
  </ul>
  <div id="tabs2-1">
    <p>Tab1 内容</p>
  </div>
  <div id="tabs2-2">
    <p>Tab2 内容</p>
  </div>
  <div id="tabs2-3">
    <p>Tab3 内容</p>
  </div>
</div>
```

使用 AJAX 可以不指定内容容器，但是也可以将 AJAX 内容放入指定容器中。

```
<li><a href="hello/world.html" title="Todo Overview"> ... </a></li>
<div id="Todo_Overview"> ... </div>
```

## 12.4.3 活用事件

Tab 控件有很多事件：

```
select, load, show, add, remove, enable, disable
```

使用这些事件可以完成很多复杂任务。需要注意事件的签名：

```
$('#example').bind('tabsselect', function(event, ui) {
    ui.tab          ui.panel
    ui.index
});
```

第一个是事件对象，第二个 ui 对象是传递的额外参数，通过 ui 对象可以获取到 Tab 对象、Tab 对象所在容器和 Tab 的索引值。

比如可以在事件中做验证：

```
$('#example').tabs({
    select: function(event, ui) {
        var isValid = ... //根据验证结果返回 true 或 false
        return isValid;
    }
});
```

或者当添加一个 tab 时立刻切换到选中状态：



```
var $tabs = $('#example').tabs({
    add: function(event, ui) {
        $tabs.tabs('select', '#' + ui.panel.id);
    }
});
```

活学活用，更多应用大家也可以参见 Tab 组件的官方文档：<http://jqueryui.com/demos/tabs>。

## 12.5 Accordion 手风琴菜单控件

使用 jQuery UI 的 Accordion 手风琴菜单控件可以实现手风琴菜单的效果，如图 12-18 所示。

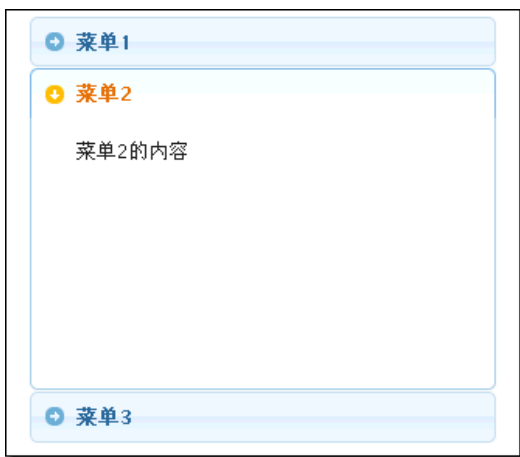


图 12-18 手风琴菜单效果

Accordion 手风琴控件文档地址为：<http://jqueryui.com/demos/accordion/>。

### 12.5.1 应用实例

使用 Accordion 控件与 tabs 控件一样，要注意 HTML 的格式。一定要提供一组头部和内容的元素，才能生成一个完整的手风琴菜单。

```
<div id="accordion">
  <h3><a href="#">First header</a></h3>
  <div>First content</div>
  <h3><a href="#">Second header</a></h3>
  <div>Second content</div>
</div>
```

作为菜单的 header 元素是可以通过设置 options 属性改变的：

```
$( ".selector" ).accordion({ header: 'h3' });
```

但是一定要保证，每一个内容元素都要紧跟在 header 元素后面。

首先通过一个实例来上手 Accordion 控件。实例的完整代码请在以下路径中寻找：

【程序代码：[jQueryStorm.Web/chapter12/Demo-UI-Accordion.htm](http://jQueryStorm.Web/chapter12/Demo-UI-Accordion.htm)】。

本实例包含三个 Accordion 控件，分别用来演示控件的不同行为。下面是即将应用

Accordion 控件的 HTML 元素片段。

```
<!-- Demo. 默认配置的 Accordion 菜单 -->
<div style="width: 300px; float:left; margin-left:20px;">
  <div id="accordion1">
    <h3><a href="#">菜单 1</a></h3>
    <div>
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
    </div>
    <h3><a href="#">菜单 2</a></h3>
    <div>
      菜单 2 的内容
    </div>
    <h3><a href="#">菜单 3</a></h3>
    <div>
      菜单 3 的内容
    </div>
  </div>
</div>
<!-- Demo. 取消自动高度, 可折叠 -->
<div style="width: 300px; float: left; margin-left: 20px;">
  <div id="accordion2">
    <h3><a href="#">菜单 1</a></h3>
    <div>
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
    </div>
    <h3><a href="#">菜单 2</a></h3>
    <div>
      菜单 2 的内容
    </div>
    <h3><a href="#">菜单 3</a></h3>
    <div>
      菜单 3 的内容
    </div>
  </div>
</div>
<!-- Demo. 鼠标滑动触发自定义图标 -->
<div style="width: 300px; float: left; margin-left: 20px;">
  <div id="accordion3">
    <div><a href="#">菜单 1</a></div>
    <div>
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
      菜单 1 的内容<br />
    </div>
  </div>
</div>
```



```

        菜单 1 的内容<br />
    </div>
    <div><a href="#">菜单 2</a></div>
    <div>
        菜单 2 的内容
    </div>
    <div><a href="#">菜单 3</a></div>
    <div>
        菜单 3 的内容
    </div>
</div>
</div>

```

使用 Accordion 控件同样要注意 HTML 代码的结构问题。稍后会详细讲解。下面接着使用 Accordion 控件：

```

<script type="text/javascript">
    var thisPage = {
        initialize: function () { //加载时执行
            //默认配置的 Accordion 菜单
            $("#accordion1").accordion({ header: "h3" });
            //取消自动高度，可折叠
            $("#accordion2").accordion({
                autoHeight: false,
                collapsible: true
            });
            //鼠标滑动触发，自定义图标
            $("#accordion3").accordion({
                icons: {
                    header: "ui-icon-circle-arrow-e",
                    headerSelected: "ui-icon-circle-arrow-s"
                },
                event: "mouseover"
            });
        }
    }
    $(thisPage.initialize());
</script>

```

第一个手风琴菜单全部采用了默认值，在单击时切换菜单，如图 12-19 所示。



图 12-19 默认手风琴菜单效果

第二个手风琴菜单，单击时可以伸缩/展开菜单项，如图 12-20 所示。





第三个手风琴菜单，在鼠标滑动时即切换菜单，注意，菜单前面的箭头也被修改了，如图 12-21 所示。

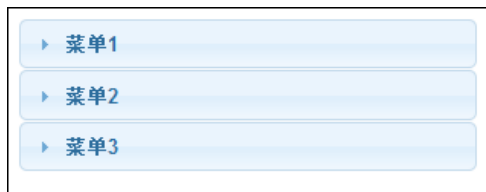


图 12-20 手风琴菜单折叠效果



图 12-21 手风琴菜单滑动切换效果

## 12.5.2 关键点讲解

### 1. 注意高度设置过小问题

当包含 Accordion 控件的容器高度设计过小时，在 Firefox 3 中在此容器后面的内容会被 Accordion 控件部分遮盖。在 IE 中没有此问题。经检查是因为容器高度小于菜单高度导致。所以在应用时应当注意不要将容器高度设置过小。

### 2. 部分关键属性

☐ **autoHeight**: 设置是否自动将内容高度设置为容器高度。

☐ **collapsible**: 设置是否可折叠。

一般上面两个配合使用，以为折叠后肯定会改变菜单高度，会导致 **autoHeight** 设置为 true 无效。

### 3. 如果想一次打开多个菜单内容，则不能使用 Accordion 控件

一个 Accordion 控件不允许多余一个菜单的打开（可以全部折叠，但是最多只能打开一个）。如果想实现一次打开多个菜单的功能，可以使用下面的语句实现：

```
jQuery(document).ready(function(){
    $('.accordion .head').click(function() {
        $(this).next().toggle();
        return false;
    }).next().hide();
});
```

还可以加入些动画效果：

```
jQuery(document).ready(function(){
    $('.accordion .head').click(function() {
        $(this).next().toggle('slow');
        return false;
    }).next().hide();
});
```



## 12.6 Progressbar 进度条控件

Progressbar 进度条控件用来显示一个任务完成的百分比，效果如图 12-22 所示。

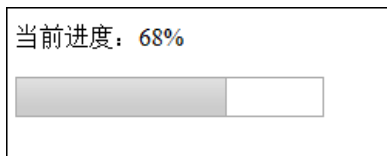


图 12-22 Progressbar 效果

Progressbar 控件本身十分简单，为其设置 value 属性来控制进度条的显示，同时还可以获取到进度条的 value 值。在一些网页游戏中，可以用来做任务完成进度的显示，或者用来做页面加载进度的显示。

有关 Progressbar 的官方文档，请参见 <http://jqueryui.com/demos/progressbar>。

### 12.6.1 应用实例

Progressbar 控件需要应用在一个 div 上，可以为这个 div 设置宽度。

```
<div>
  <p>当前进度: <span id="pValue">Loading...</span>%</p>
  <div id="divProgressbar" style="width:200px;"></div>
</div>
```

也可以为它的父类设置宽度，则 Progressbar 控件将自动适应它的父类元素。

```
<div style="width:200px;">
  <p>当前进度: <span id="pValue">Loading...</span>%</p>
  <div id="divProgressbar"></div>
</div>
```

Progressbar 的 options 属性只有两个，一个是“disabled”用来控制是否有效，另一个“value”用来控制显示进度。

本实例通过使用队列实现一个进度条从 0~100 加载过程的效果，下面是实例的完整代码：

```
【代码路径：jQueryStorm.Web/chapter12/ Demo-UI-Progressbar.htm】
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery UI - progressbar 进度条控件应用实例</title>
  <link href="../../Static/common/css/Smoothness/style.css" rel="stylesheet" type="text/css" />
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
  <script src="../../Static/common/js/jquery-ui-1.8.2.custom.min.js" type="text/javascript"></script>
</head>
<body>
  <div style="width:200px;">
    <p>当前进度: <span id="pValue">Loading...</span>%</p>
    <div id="divProgressbar"></div>
  </div>
  <script type="text/javascript">
    var thisPage = {
      initialize: function () { //加载时执行
        this.pbar = $("#divProgressbar").progressbar({
```

```

        value: 0,
        change: function (event, ui) { $("#pValue").html(thisPage.pbar.progressbar("value")); }
    });

    for (var i = 0; i < 100; i++) {
        $("#divProgressbar").queue("myQueue", function (next) {
            thisPage.pbar.progressbar("value", thisPage.pbar.progressbar("value") + 1);
            next();
        });
        $("#divProgressbar").delay(50, "myQueue");
    }

    $("#divProgressbar").dequeue("myQueue");
}
$(thisPage.initialize());
</script>
</body>
</html>

```

效果如图 12-23 所示。

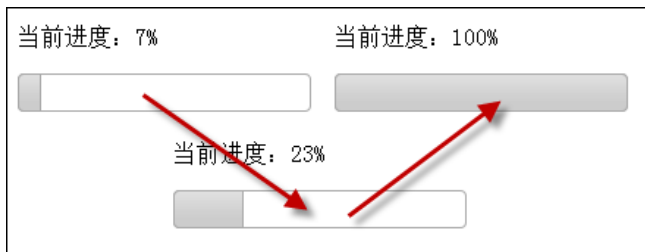


图 12-23 Progressbar 队列加载效果

## 12.6.2 实例讲解

图 12-23 的实例是使用队列实现了每 50 毫秒改变一次进度条的进度值，并将进度条的数值显示在一个 span 中。

在讲解 jQuery 动画的时候已经讲了队列的使用，但是那时使用队列是为了和动画打交道，使用的是“fx”这个动画队列。在本示例中，使用了自定义队列“myQueue”。使用“.queue()”函数为 myQueue 队列添加方法：

```

    for (var i = 0; i < 100; i++) {
        $("#divProgressbar").queue("myQueue", function (next) {
            thisPage.pbar.progressbar("value", thisPage.pbar.progressbar("value") + 1);
            next();
        });
        $("#divProgressbar").delay(50, "myQueue");
    }

```

为了让函数的执行延时，所以使用 delay 函数设置了 50 毫秒的延时。

最后使用 dequeue() 函数来让队列中的第一个函数执行。因为在每个函数中都使用了 next() 来执行下一个函数，所以实现的效果就是队列中的函数逐个依次执行：

```

$("#divProgressbar").dequeue("myQueue");

```

在下列函数中，操作了 Progressbar 对象的 value 值。通过设置 Progressbar 的 change 事件，在 value 值改变的时候，将 value 值显示在 span 元素上：



```
change: function (event, ui) { $("#pValue").html(thisPage.pbar.progressbar("value")); }
```

## 12.7 Slider 滑动条控件

Slider 滑动条控件的效果，如图 12-24 所示。

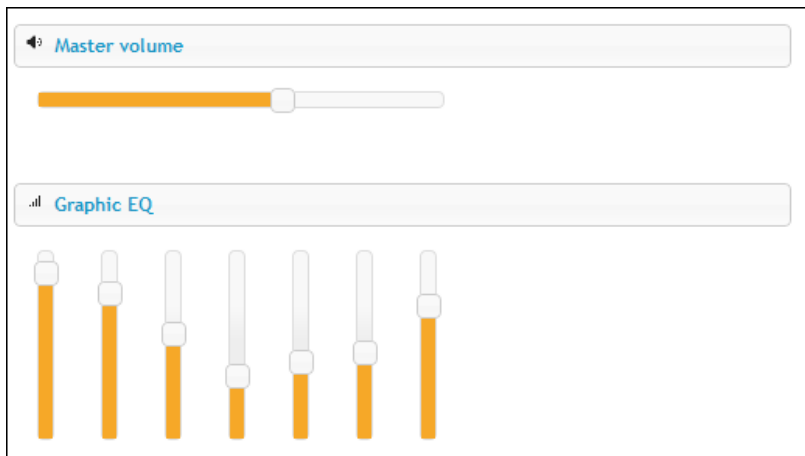


图 12-24 Slider 滑动条控件

Slider 控件是作用在一个 div 上，可以将一个 div 元素变成可以滑动的效果。通常用在范围筛选，比如价格筛选等场景。

Slider 控件的官方文档，请参见 <http://jqueryui.com/demos/slider/>。

### 12.7.1 应用实例

本节通过实例，介绍几种 Slider 控件的常见用法。下面是实例的完整代码：

【代码路径：jQueryStorm.Web/chapter12/ Demo-UI-Slider.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery UI - Slider 控件应用实例</title>
  <link href="../../Static/common/css/Smoothness/style.css" rel="stylesheet" type="text/css" />
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
  <script src="../../Static/common/js/jquery-ui-1.8.2.custom.min.js" type="text/javascript"></script>
</head>
<body>
  <div style="width:200px;">
    <p>默认设置：<span id="value1">0</span></p>
    <div id="divSlider1"></div>

    <p>设置 50 滑动间隔：<span id="value2">100</span></p>
    <div id="divSlider2"></div>

    <p>范围选择：<span id="value3-1">0</span> - <span id="value3-2">500</span></p>
    <div id="divSlider3"></div>

    <p>垂直的滑动条：<span id="value4">38</span></p>
    <div id="divSlider4"></div>
```



```

</div>
<script type="text/javascript">
  var thisPage = {
    initialize: function () { //加载时执行
      $("#divSlider1").slider({
        slide: function (event, ui) { $("#value1").html(ui.value) }
      });

      $("#divSlider2").slider({
        slide: function (event, ui) { $("#value2").html(ui.value) },
        value: 100,
        min: 0,
        max: 500,
        step: 50
      });

      $("#divSlider3").slider({
        slide: function (event, ui) {
          $("#value3-1").html(ui.values[0]);
          $("#value3-2").html(ui.values[1]);
        },
        range: true,
        values: [0, 500],
        min: 0,
        max: 500
      });

      $("#divSlider4").slider({
        slide: function (event, ui) {
          $("#value4").html(ui.value);
        },
        range: "min",
        value: 38,
        orientation: "vertical"
      });
    }
  }

  $(thisPage.initialize());
</script>
</body>
</html>

```

本实例演示了几种 Slider 控件的用法，首先是默认设置的效果，如图 12-25 所示。

默认情况下，可以选择 0~100 中的任意一个值。可以设置两个值之间的间隔至少是 50，效果如图 12-26 所示。

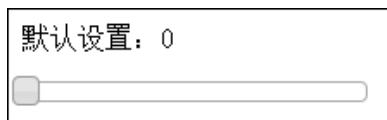


图 12-25 Slider 控件默认设置

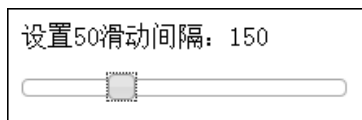


图 12-26 Slider 设置间隔

还可以将控件设置成选择范围，即拥有两个滑动条，如图 12-27 所示。

滑动条不仅仅可以水平放置，还可以垂直放置，如图 12-28 所示。



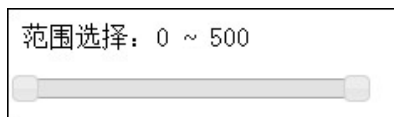


图 12-27 Slider 选择范围



图 12-28 垂直放置的 Slider 控件

### 12.7.2 实例讲解

Slider 滑动条控件可以有一个滑动块，也可以有两个。在这两种使用场景中，一个使用 `value` 属性设置和获取值，另一个使用 `values` 属性设置和获取值。

在 12.7.1 节的实例中，为了能将滑动条的值显示在页面上，修改了 `Slide` 事件：

```
slide: function (event, ui) { $("#value1").html(ui.value) }
```

这里需要注意的是 `slide` 事件的签名。其中 `event` 是 jQuery 格式化后的事件对象。`ui` 是一个对象，具有以下三个属性。

- ❑ `ui.handle`：DOM 元素，当前用于滑动的滑动块，通常是一个 `a` 元素。
- ❑ `ui.value`：数字。如果是单滑动块，则通过此属性获取选择的值。在范围选择时此属性能获取到最小值。
- ❑ `ui.values`：数组，如果是范围选择，则此数组[0]表示起始值，[1]表示结束值。

如果是单滑动块，则没有 `ui.values` 属性。在有两个滑动块时，才同时具有这三个属性，其中 `ui.value` 是所选范围的最小值。

## 12.8 button 按钮控件

`button` 是在新版本的 jQuery UI 中加入的控件。`button` 控件能够将表单元素、各种类型的 `input` 元素（比如 `submit`）、`a` 元素等变成按钮样式显示，并且具有 `mouseover` 等样式效果。

### 12.8.1 应用实例

首先通过一个例子，看一下 `button` 控件的效果，随后再进行详细的讲解。

【代码路径：jQueryStorm.Web/chapter12/ Demo-UI-Button.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery UI - Button 控件应用实例</title>
  <link href="../../Static/common/css/Smoothness/style.css" rel="stylesheet" type="text/css" />
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
  <script src="../../Static/common/js/jquery-ui-1.8.2.custom.min.js" type="text/javascript"></script>
</head>
<body>
  <div id="simple">
    <button>
```



```

        button 元素</button>
        <input type="submit" value="submit 类型的 input 元素" />
        <a href="http://www.elong.com">"a" 元素</a>
    </div>
    <br />
    <div id="radio">
        <input type="radio" id="radio1" name="radio" /><label for="radio1">单选框 1</label>
        <input type="radio" id="radio2" name="radio" checked="checked" /><label for="radio2">单选框
2</label>
        <input type="radio" id="radio3" name="radio" /><label for="radio3">单选框 3</label>
    </div>
    <br />
    <div id="check">
        <input type="checkbox" id="check1" /><label for="check1">复选框 A</label>
        <input type="checkbox" id="check2" /><label for="check2">复选框 B</label>
        <input type="checkbox" id="check3" /><label for="check3">复选框 C</label>
    </div>
    <br />
    <div class="demo">
        <button id="beginning">
            go to beginning</button>
        <button id="rewind">
            rewind</button>
        <button id="play">
            play</button>
        <button id="stop">
            stop</button>
        <button id="forward">
            fast forward</button>
        <button id="end">
            go to end</button>
        <input type="checkbox" id="shuffle" /><label for="shuffle">Shuffle</label>
        <span id="repeat">
            <input type="radio" id="repeat0" name="repeat" checked="checked" /><label
for="repeat0">No Repeat</label>
            <input type="radio" id="repeat1" name="repeat" /><label for="repeat1">Once</label>
            <input type="radio" id="repeatall" name="repeat" /><label for="repeatall">All</label>
        </span>
    </div>
    <script type="text/javascript">

        var thisPage = {
            initialize: function () { //加载时执行
                $("#simple button, #simple input, #simple a").button();
                $("a").click(function () { return false; });
                $("#radio").buttonset();
                $("#check").buttonset();

                $('#beginning').button({
                    text: false,
                    icons: {
                        primary: 'ui-icon-seek-start'
                    }
                });
                $('#rewind').button({
                    text: false,
                    icons: {
                        primary: 'ui-icon-seek-prev'
                    }
                });
                $('#play').button({
                    text: false,

```



```

        icons: {
            primary: 'ui-icon-play'
        }
    }).click(function () {
        var options;
        if ($(this).text() == 'play') {
            options = {
                label: 'pause',
                icons: {
                    primary: 'ui-icon-pause'
                }
            };
        } else {
            options = {
                label: 'play',
                icons: {
                    primary: 'ui-icon-play'
                }
            };
        }
        $(this).button('option', options);
    });

    $('#stop').button({
        text: false,
        icons: {
            primary: 'ui-icon-stop'
        }
    }).click(function () {
        $('#play').button('option', {
            label: 'play',
            icons: {
                primary: 'ui-icon-play'
            }
        });
    });

    $('#forward').button({
        text: false,
        icons: {
            primary: 'ui-icon-seek-next'
        }
    });

    $('#end').button({
        text: false,
        icons: {
            primary: 'ui-icon-seek-end'
        }
    });
    $("#shuffle").button();
    $("#repeat").buttonset();

    }

    $(thisPage.initialize());

```

```

</script>
</body>
</html>

```

效果如图 12-29 所示。





图 12-29 button 控件显示效果

在这里实例中，分别将 button 元素、input 元素、a 元素、单选框、复选框应用了 button 控件。最后用一个播放器综合示例。音乐网站上面的播放器就可以如此做。

## 12.8.2 实例讲解

使用 button 控件最简单的用法，是直接使用 button()方法：

```
$("#simple button,#simple input,#simple a").button();
```

实例中的 button 元素、input 元素、a 元素都是使用 button()方法实现的。

对于 a 超链接元素需要注意的是，因为 a 元素的浏览器默认行为是跳转到“href”属性设置的页面，所以要取消 a 元素的这种默认行为，可以使用下面的两种方式：

```
$("#a").click(function () { return false; });
$("#a").click(function (e) { e.preventDefault(); });
```

在 a 元素的 click 事件内返回 false 或者调用事件对象的 preventDefault()方法，都可以取消 a 元素的默认行为。

对于单选框、复选框这种成组的元素，需要使用 buttonset()方法。

```
$("#radio").buttonset();
$("#check").buttonset();
```

其中的 radio 和 check，是包含一组 input 元素的外层 div 的 ID 值。

```
<div id="radio">
<input type="radio" id="radio1" name="radio" /><label for="radio1">单选框 1</label>
<input type="radio" id="radio2" name="radio" checked="checked" /><label for="radio3">单选框 2</label>
<input type="radio" id="radio3" name="radio" /><label for="radio2">单选框 3</label>
</div>
```

对于单选框和复选框，每个 input 元素需要对应一个 label 元素，在最后显示时，显示在页面上的按钮实际上是 label 对象。但是当单击 label 时会改变对应的 input 对象。如果没有对应的 label 元素，这个 input 控件也不会显示。默认情况下按钮显示的文字就是 label 对象的文字，但是可以通过修改 label 参数来修改文字内容：

```
$("#radio2").button("option", "label", "修改后的 radio2");
```

button()和 buttonset()是 button 控件最基本的两个函数。要销毁 button 控件，可以使用 button("destroy")方法：

```
$("#simple button,#simple input,#simple a").button("destroy");
```





```
$("#radio input").button("destroy");
```

在最后播放器的例子中，使用了 `icon` 参数。`icon` 参数用来为按钮显示图标。只有当设置了 `icon` 时，`text` 属性才会生效。`text` 属性是 Boolean 值，`false` 表示只显示图标不显示文字。比如 `stop` 按钮：

```
<button id="stop">stop</button>
```

设置它只显示图标，不显示文字：

```
$('#stop').button({
  text: false,
  icons: {
    primary: 'ui-icon-stop'
  });
```

使用 `button` 控件最主要的问题就是样式与脚本行为的耦合。只用当 `button()` 生效后，才能显示出按钮的样式。所以如果页面脚本都放在最后加载，将会导致在脚本执行时，页面按钮样式“突然改变”。因此在企业项目中，并不推荐使用此控件，一般的页面设计师都能制作出更好看易用的按钮对象。

想查看更全面的 `button` 控件的信息，请参阅官方文档 <http://jqueryui.com/demos/button/>。

## 12.9 autocomplete 自动提示控件

在 jQuery 插件一章，讲解了一个名为“autocomplete”的插件。jQuery UI 中提供了一个同名的控件，也叫做“autocomplete”。此控件是在新版本的 jQuery UI 中加入的。

jQuery UI 的 `autocomplete` 的优点就是官方原生，和 jQuery 的皮肤完美集成。缺点是不够灵活，功能稍显不足。

但是对于简单的应用而言，jQuery UI 的 `autocomplete` 控件也够用了。本节下文所述的 `autocomplete` 控件泛指 jQuery UI 中的控件。接下来通过实例讲解 `autocomplete` 控件的使用。

### 12.9.1 应用实例

本实例是一个出发城市的输入框。在此输入框上应用 `autocomplete` 插件。

【代码路径：jQueryStorm.Web/chapter12/ Demo-UI-Autocomplete.htm】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery UI - Autocomplete 控件应用实例</title>
  <link href="../../Static/common/css/Smoothness/style.css" rel="stylesheet" type="text/css" />
  <script src="../../Static/common/js/jquery-1.4.2.min.js" type="text/javascript"></script>
  <script src="../../Static/common/js/jquery-ui-1.8.2.custom.min.js" type="text/javascript"></script>
</head>
<body>
  <div>
    <label>
      出发城市:
    </label>
    <input id="startCity" />
    <input id="hStartCity" name="startCity" type="hidden" />
```



```

</div>
<script type="text/javascript">
    var thisPage = {
        initialize: function () {
            this.$citys = [{ label: "Beijing 北京", value: "北京", cityId: "BJS" }, { label: "Shanghai
上海", value: "上海", cityId: "SHA" }];
            this.$startCity = $("#startCity");
            this.$hStartCity = $("#hStartCity");
            //对 startCity 输入框应用 autocomplete 控件
            this.$startCity.autocomplete({
                source: this.$citys,
                select: function (event, ui) {
                    jQuery("#startCity").val(ui.item.value);
                    jQuery("#hStartCity").val(ui.item.cityId);
                    jQuery("#endCity").click();
                },
                minLength: 0
            });

            this.$startCity.click(jQuery.proxy(this.onStartCityFocus, this));

            onStartCityFocus: function (e) {
                if ($(e.target).val() === "") {
                    this.$startCity.autocomplete("search", "");
                }
            }
        }
    }

    $(function () {
        thisPage.initialize();
    })
</script>
</body>
</html>

```

本实例实现了两个功能。一是在 `input` 中输入字符时，比如输入“bei”，会自动提示“北京”。二是当 `input` 内容为空并单击时，会同时显示“北京”、“上海”这两个城市。本实例的数据源只有北京和上海，如果添加更多的数据源则会显示更多的城市。

效果如图 12-30 和图 12-31 所示。



图 12-30 单击出发城市输入框效果



图 12-31 输入 bei 自动提示效果

## 12.9.2 实例讲解

要使用 `autocomplete` 插件，首先要准备好数据源。本例中准备了一个简单的数组作为数据源：

```

this.$citys = [{ label: "Beijing 北京", value: "北京", cityId: "BJS" }, { label: "Shanghai 上海", value: "上海", cityId: "SHA" }];

```

其中数组中的每一个元素都是一个 `map` 对象。



map 对象的 label 属性是自动提示时显示的文字。value 是当选中时向 input 元素填充的内容。另外在系统搜索中实际使用的是 cityId，所以还准备了 cityId 属性用于填充隐藏域。

label 和 value 这两个属性名的强制规则就是 autocomplete 控件最不灵活的地方。因为一个标准的“数据源”的结构应该是：城市中文名、城市英文名等。而 label 属性其实仅仅是一个用于显示的字段，value 是一个用于填充内容的字段，类似于 MVC 中的 ViewModel 即视图模型，是用于显示的模型类。而 jQuery 插件一章中提到的 autocomplete 插件，其设计思想就要好很多。可以定义一个任意结构的标准数据模型，然后定义显示时使用哪些字段、选中时填充哪些字段、搜索时填充哪些字段。相信这种设计思想是值得 jQuery UI 学习的。

autocomplete 的数据源除了是数组，还可以是 string 或者 function。

如果是 string，会把这个 string 当做 URL 处理，自动从此 URL 获取数据源。比如：

```
$("#birds").autocomplete({
    source: "search.php",
    minLength: 2,
    select: function(event, ui) {
        log(ui.item ? ("Selected: " + ui.item.value + " aka " + ui.item.id) : "Nothing selected, input was "
+ this.value);
    }
});
```

autocomplete 支持 JSONP，所以可以使用一个跨域的地址，但是一定要保证此地址也支持 JSONP。

如果数据源是 function，则用起来最麻烦但是灵活性最大。比如：

```
source: function(request, response) {
    $.ajax({
        url: "http://ws.geonames.org/searchJSON",
        dataType: "jsonp",
        data: {
            featureClass: "P",
            style: "full",
            maxRows: 12,
            name_startsWith: request.term
        },
        success: function(data) {
            response($.map(data.geonames, function(item) {
                return {
                    label: item.name + (item.adminName1 ? ", " + item.adminName1 :
                    "")) + ", " + item.countryName,
                    value: item.name
                }
            })))
        }
    })
},
```

首先来看 function() 函数的签名：function(request, response)。

request 是一个对象，仅包含一个 term 属性，即用户输入的文字。服务器端需要根据此值返回搜索后的数据。

response() 是一个函数，只接受一个参数，即 autocomplete 期望的数据源。上面的例子中，首先把返回的 data 对象，转换成了含有 label 和 value 属性的对象，然后调用 response() 函数并传入此对象。一旦调用了 response() 函数，自动建议功能就会被触发。



autocomplete 同样通过一个 options 参数对象设置其各种属性，其中数据源 source 就是 options 参数的一个属性。除了 source，还有下面几个关键的可选参数。

- ❑ disabled: 默认为 false，控件是否有效。
- ❑ delay: 数字值，表示延迟多少毫秒开始执行查询，默认是 300（毫秒）。如果用户连续输入并且建个小于 delay 设定的值，则只有当用户停止输入时才会做一次查询。delay 的值很有必要，可如果想在 Google 首页自动提示，用户每次敲击键盘都进行一次自动提示，服务器的压力和网络流量都将成倍地增大。
- ❑ minLength: 数字，表示用户至少输入几个字符就进行自动提示，默认是 1。

本节实例中，为 input 元素绑定了一个 click 事件：

```
onStartCityFocus: function (e) {  
    if ($(e.target).val() === "") {  
        this.$startCity.autocomplete("search", "");  
    }  
}
```

在此事件中，触发了控件的 search 即搜索功能，并且传入的是一个空字符串，所以要想让此事件被正确触发，就要设置 minLength 属性为 0。

autocomplete 提供了若干的事件和方法。有关这些事件和方法的详细功能，请参见官方文档 <http://jqueryui.com/demos/autocomplete>。

## 12.10 小结

本章对 jQuery UI 的所有控件（Widgets）做了讲解。虽然这些控件非常实用，但是切记 jQuery UI 不仅仅是这几个控件，jQuery UI 是一个页面 UI 库，除了提供现成的控件，还提供了若干个用户交互接口。比如让图层可拖动、可拉伸等。如果不想使用 jQuery UI 中的控件，完全可以使用这些交互接口自己开发控件。

对于大部分个人网站和简单应用类网站，使用 jQuery UI 中提供的控件将极大地加快开发效率。甚至可以整站使用 jQuery UI 的皮肤，这样连美工的工作都会轻松许多。但是 jQuery UI 的控件也存在很多问题。并不是适用于所有的站点。尤其是更复杂、要求更高的站点。

在第 13 章中，将讲解如何基于 jQuery 打造脚本框架和控件库。





## 第 13 章



# 基于 jQuery 打造脚本框架

无论是国内还是国外,总有人喜欢拿 jQuery 与其他的脚本框架相比较,比如 Mootools、ExtJS 等。推崇其他脚本库框架的人,无非就是几个理由:更健全的框架、更多的功能、更好的编程体系等。这说明他们没有真正地理解 jQuery。jQuery 不是拿来直接作为脚本框架的,而是用来打造框架的。jQuery 的高效性、易用性、简单性,让其具有了成为一个系统脚本框架“基石”的各种特质。

简单的系统,可以直接使用 jQuery 类库,而不需要脚本框架。复杂系统,视系统规模开发各种重量级的脚本框架。脚本框架都是“通用”的,但是并不一定都“适用”。

本章将基于 jQuery,打造一个大型网站的脚本框架。这比直接使用 Mootools 更简单、更容易维护、更易用。

## 13.1 页面脚本管理

JavaScript 程序如果不加以组织和管理，常常导致复用性低，并且分散在系统的各个角落。所以可以使用面向对象的设计方式，为每个页面创建一个脚本对象，用管理对象的方式管理脚本文件。

### 13.1.1 使用面向对象的方式管理页面脚本

在 ASP.NET 程序中，一个页面就是一个类。用同样的方式管理脚本程序，首先需要为一个页面创建一个脚本对象，比如对于一个名称为“HomePage”的页面，为其创建一个同名的对象：

```
var HomePage = function () {  
}
```

HotelSeachBox()是一个函数，这个函数相当于类的构造函数。

对于网页，可以将脚本的全局事件抽象成以下几个：

- ☐ 初始化 DOM 元素。
- ☐ 事件绑定。
- ☐ 页面加载。

关于这些事件的具体作用，将在后面详细讲解，本节只关注如何使用面向对象的方式创建脚本对象。接下来将这些事件放在“HomePage”的构造函数中，就可以按照顺序执行了：

```
var HomePage = function () {  
    this.initializeDOM(); //初始化 DOM  
    this.initializeEvent(); //绑定事件  
    this.initialize(); //页面初始化  
}
```

在使用 new 运算符创建此类的一个实例之前，还需要创建这几个事件的事件处理函数，否则在 new 时会提示找不到函数：

```
HomePage.prototype.initializeDOM = function(){};  
HomePage.prototype.initializeEvent = function(){};  
HomePage.prototype.initialize = function(){};
```

在每个事件中，可以编写响应的代码。比如 initializeDOM 中编写获取 jQuery 对象的代码。这些事件将在后面讲解。现在使用 new 运算符创建类的一个实例时，会自动运行构造函数，分别运行这些事件。

显然上面的代码不够优美，对于每一个页面，都要创建一个这样一个类。所以使用一个工厂方法进行封装：

```
var Class = {  
    create: function (classObj)  
    {  
        /// <summary>创建类的函数</summary>  
        /// <param name="classObj" type="object">用对象表示的类</param>  
        /// <returns type="object">类的实例，已经执行了类的构造函数</returns>  
        var args = jQuery.makeArray(arguments);
```



```

args.shift();
var tempclassObj = function (params)
{
    this.initialize.apply(this, params);
    this.initializeDOM.apply(this, params);
    this.initializeEvent.apply(this, params);
    this.pageLoad.apply(this, params);
};
classObj.initialize = classObj.initialize || jQuery.noop;
classObj.initializeDOM = classObj.initializeDOM || jQuery.noop;
classObj.initializeEvent = classObj.initializeEvent || jQuery.noop;
classObj.pageLoad = classObj.pageLoad || jQuery.noop;
tempclassObj.prototype = classObj;
var result = new tempclassObj(args);
return result;
}
};

```

“Class.create()”函数用于协助创建一个类的实例。在 JavaScript 中，通常使用面向对象的方式创建类，都是使用一个 function 作为构造函数，然后扩展它的原型。为了能使用 JSON 对象定义一个类，“Class.create”函数内部使用了这个技巧，创建了一个函数“tempclassObj()”作为构造函数的委托，而函数内部实际上是依次调用类对象的“initialize”、“initializeDOM”等方法。也就是说，真正的构造函数是“initialize”、“initializeDOM”等这些方法。通过扩展函数“tempclassObj”的原型，构建了一个 JavaScript 中的“类”。然后使用“new”运算符创建一个类实例。有关 JavaScript 的面向对象知识，可以参考第 2 章。

使用 Class.create()方法改造后的代码如下：

```

var HomePage = {
    initializeDOM: function () {
        this.$btnSearch = jQuery("#btnSearch");
    },
    initializeEvent: function () {
        this.$btnSearch.click(function () { alert(this.id + " clicked!"); });
    },
    pageLoad: function () {
        alert("page load");
    }
}
var homePage = Class.create(HomePage);

```

homePage 对象是使用 Class.create 创建的一个 HomePage 类的实例。在创建的时候会分别触发 initlize 构造函数、初始化 DOM、绑定事件、页面加载这些事件。这里声明一个类，使用的是标准的 JSON 格式，所以对象结构十分清晰。这几个事件即使在类中没有定义，比如没有在 HomePage 类中定义，initlize()方法也不会报错，因为在 Class.create()函数中做了处理，会默认提供一个空函数。

类的命名使用 Pascal 规则，所有首字母大写，比如 HomePage 类。具体实例对象命名则使用 camel 规则，除了第一个单词首字母小写，其余单词首字母都大写。比如 new 出来的 homePage 变量。

使用 Class.create()还可以为构造函数传递参数，比如：

```

var TestPage = {
    initialize: function(msg){
        alert(msg);
    }
}

```





```
}
var testPage = Class.create("TestPage", "Hello jQuery!");
```

现在 `homePage` 变量就是 `HomePage` 类的一个实例。接下来学习如何使用抽象出的这几个页面事件。

### 13.1.2 页面脚本事件

在 13.1.1 中，已经将一个页面的脚本加载，分成了 5 个事件。

#### 1. 构造函数: initialize

`initialize` 是作为类的构造函数存在的。所以最先被执行。在 `initialize` 中，可以用来初始化对象的各个属性。通常有时候某些变量需要在服务器端输出到页面上，比如服务器端在页面上生成了一个 `orderId` 对象，就可以在 `initialize()` 方法中使用将 `orderId` 复制到对象的属性中来：

```
initialize: function(){
    this.orderId = window.orderId;
}
```

#### 2. 初始化 DOM 元素: initializeDOM

在页面上，常常要对一个对象多次操作。如果每次操作都是用 jQuery 选择器重新获取，虽然 jQuery 选择器效率很高，但是仍然会花费时间和资源。所以在 `initializeDOM` 事件中，将所有需要操作的页面对象都预先地获取，并保存在属性变量中，这样就可以重复使用。比如页面上的一个按钮：

```
this.$bthSearch = jQuery("#btnSearch");
```

使用 ID 选择器，将 ID 为 `btnSearch` 的按钮对象，保存在了页面脚本对象的 `$bthSearch` 属性中。在事件绑定时不需要再获取对象，可以直接使用 `$bthSearch` 属性：

```
this.$bthSearch.click(function () { alert(this.id + " clicked!"); });
```

以 `$` 开头命名，是为了说明这是一个 jQuery 对象而不是 DOM 对象。这是一种使用习惯，当然也可以在你的系统中使用“`jq`”或任意的字母替换。

#### 3. 事件绑定: initializeEvent

在事件绑定中，为页面上的指定元素添加事件，比如为按钮添加单击事件，在 `initializeDOM` 事件中已经获取到了元素，所以在 `initializeEvent` 中可以直接使用：

```
this.$bthSearch.click(function(){alert("clicked!");});
```

在 `initializeEvent` 事件中，`this` 是指页面脚本对象，所以可以通过“`this.$btnSearch`”访问到对象的 `$btnSearch` 属性，同样可以将事件处理函数也作为页面脚本对象的一个属性，然后通过 `this` 访问：

```
var HomePage = {
    //省略部分代码
    initializeEvent: function () {
        this.$bthSearch.click(this.onBthSearchClick);
    },
    onBthSearchClick: function()
```



```

    {
        alert("clicked!");
    }
}

```

上面的代码可以正确地对象绑定事件。但是要特别注意，在事件处理函数中，“this”指针并不是页面脚本对象。下面的代码，btnSearch 单击时会产生错误：

```

var HomePage = {
    initializeDOM: function () {
        this.$bthSearch = jQuery("#btnSearch");
    },
    initializeEvent: function () {
        this.$bthSearch.click(this.onBthSearchClick);
    },
    onBthSearchClick: function()
    {
        alert(this.myName);
    },
    myName: "Ziqiu"
}

```

onBthSearchClick 事件通过 this 访问 HotelSearchBox 对象的 myName 属性，但是当单击时会出现错误，提示找不到 myName 属性。这是因为脚本中的 this 总是指向函数的调用者，单击事件的触发者是页面，可以理解为页面调用了 onBthSearchClick 函数，所以此时 onBthSearchClick 事件中的 this 是指页面对象。

习惯了高级语言的开发人员会对此产生疑惑。但是可以通过 jQuery 提供的 proxy() 函数解决此问题，只需要将事件绑定时的语句改为：

```

this.$bthSearch.click(jQuery.proxy(this.onBthSearchClick, this));

```

现在事件处理函数中的 this 是指将指向 HomePage 这个当前页的页面脚本对象了，可以使用此对象的任何一个属性和方法。

#### 4. 页面加载：pageLoad

pageLoad 事件类似于 Asp.Net 中的 PageLoad，在 pageLoad 中可以使用任意 DOM 对象，并且已经为各个对象完成了事件绑定。通常在页面加载时需要执行的业务逻辑代码放在 pageLoad 事件中。比如：

```

pageLoad: function () {
    this.myName += " can't fly!";
}

```

现在 HomePage.aspx 页面对应的脚本对象 HomePage 已经建好了。将所有的脚本代码放在一个单独的 HomePage.js 中。一个脚本对象存放一个 .js 文件，类似于 .NET 中的一个类存放在一个 .cs 后缀的文件中。

下面来看一下完整的页面脚本对象代码：

```

var HomePage = {
    initialize : function(){
        this.myName = "Ziqiu";
    },
    initializeDOM: function () {
        this.$bthSearch = jQuery("#btnSearch");
    },
    initializeEvent: function () {

```



```

        this.$bthSearch.click(jQuery.proxy(this.onBthSearchClick, this));
    },
    pageLoad: function () {
        this.myName += " can't fly!";
    },
    onBthSearchClick: function()
    {
        alert(this.myName);
    }
}

```

在 `initialize` 事件中，初始化了对象的 `myName` 属性。在 `initializeDOM` 事件中，获取到了页面上 ID 为 “`btnSearch`” 的按钮，并保存到了对象的 “`$bthSearch`” 属性中。

接下来在 `initializeEvent` 事件中，为按钮绑定了 `onClick` 事件。使用 `jQuery.Proxy` 改变了事件处理函数的 `this` 指针，这样在触发 “`onBthSearchClick`” 时，才可以正确地访问到 “`this.myName`” 属性。

在 `pageLoad` 事件中，再次修改了 `myName` 属性。通常 `pageLoad` 中处理一些业务逻辑。比如判断用户浏览器是否合法等。

### 13.1.3 切割脚本文件

将脚本分门别类放在独立的 `js` 文件中会便于维护，但是页面优化准则中又要求尽量减少外部的 `http` 请求数。

所以，通常规定一个页面只引用两个 `js` 文件。

一个是公共类库文件比如 `Core.js`，保存比如 `Class.create()` 这种通用函数。`Core.js` 中通常包括页面公共的脚本控件，比如 “日历框” 等。但是在脚本控件的开发过程中，通常都是一个控件一个独立的 `js` 文件。所以可以只在最后站点发布时做脚本合并。有关脚本的合并和压缩稍后就将讲到。

一个是页面脚本文件比如 `HomePage.js`，存放页面自己的脚本类比如 `HomePage` 类。在 `HomePage.js` 文件中，首先要定义 `HomePage` 类：

```

var Homepage = {
}

```

然后，在页面 DOM 加载完毕后，创建类实例。

```

$(function () {
    var homepage = Class.create(Homepage);
})

```

在页面上要按照顺序分别引用这两个 `js` 文件。但是要注意引用 `js` 文件在页面中出现的位置，为了保证能够将页面尽快地展示给用户，而页面对于 `js` 文件的加载是异步加载同步调用的。也就是说如果页面出现了 `script` 元素，就一定会等待这个脚本执行完毕才会继续加载后面的页面内容（注意这里说的是执行完毕，因为多个 `script` 块可以异步同时加载，但是一定会按照先后顺序执行）。所以在这里将所有的 `js` 文件引用都放在 `body` 的最后：

```

<body>
//忽略部分代码
<script src="../../js/Core.js" type="text/javascript"></script>
<script src="../../js/HomePage.js" type="text/javascript"></script>
</body>

```



### 13.1.4 为脚本文件添加智能提示

在编写脚本时使用智能提示可以加快开发效率。在 Visual Studio 2010 中，脚本的智能提示更加强大。

首先，如果一个脚本要使用另外一个脚本中的方法，可以在头部添加引用，比如在 HomePage.js 的头部，添加：

```
/// <reference path="../../../common/js/jquery-1.4.2.js" />
/// <reference path="../../../common/js/Core.js" />
/// <reference path="../../../common/js/TopCalendar.js" />
/// <reference path="../../../common/js/jquery-ui-1.8.2.custom.min.js" />
```

上面通过 reference 标签，引用了 jQuery 类库和若干个 js 文件。现在使用 jQuery 的方法，会出现智能提示，如图 13-1 所示。

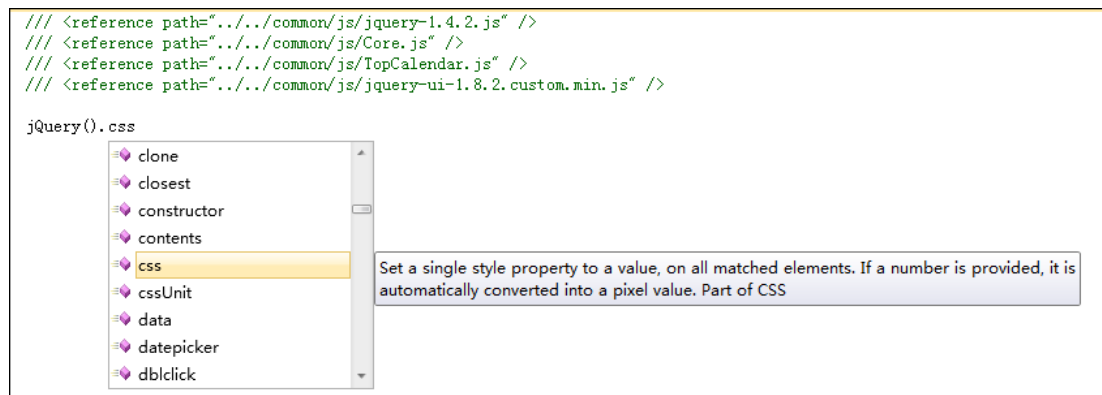


图 13-1 在 js 文件中启用智能提示

注意这里的 jQuery 注释相当详细，但是这些注释信息并不在 jQuery-1.4.2.js 这个文件中，包含完整注释的文件是 jQuery-1.4.2.js 同目录下的 jQuery-1.4.2-vsdoc.js 文件。Visual Studio 会自动去搜索是否存在此文件，如果存在则使用此文件的注释信息。这是 Visual Studio 对 jQuery 的特殊支持。

自己的类库代码，也应该像 jQuery 的注释一样，为方法和参数添加注释信息。比如“Class.create()”函数，在编写脚本方法时，也可以添加注释：

```
var Class = {
  create: function (class) {
    /// <summary>创建类的函数</summary>
    /// <param name="class" type="object">用对象表示的类</param>
    /// <returns type="object">类的实例，已经执行了类的构造函数</returns>

  }
};
```

如图 13-2 所示为在方法上添加了智能提示的效果。



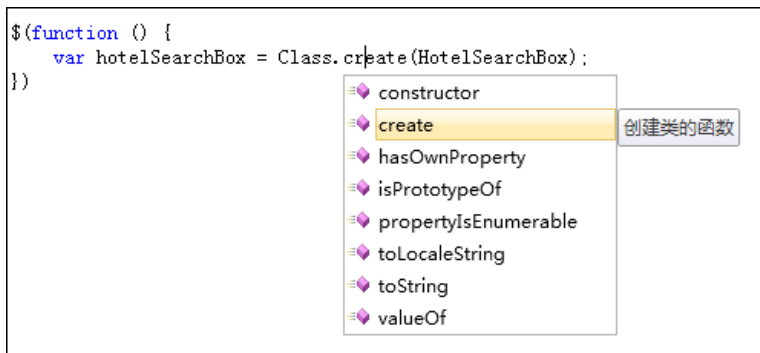


图 13-2 JavaScript 方法的智能提示效果

图 13-3 演示了在参数上添加了智能提示后的效果。

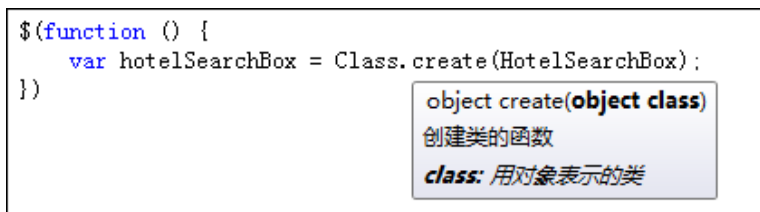


图 13-3 JavaScript 方法参数的智能提示效果

### 13.1.5 合并及压缩脚本文件

jQuery 提供了一个 .min 后缀的压缩后的类库文件，这个文件比没有压缩的文件要小很多。减小 js 文件大小可以让用户更快速地下载 js 文件。所以为了追求极致的用户体验，可以对网站项目的所有 js 文件都进行文件压缩。通常使用的工具是 YUICompressor，网址是 <http://developer.yahoo.com/yui/compressor/>。

.NET 版本 <http://yuicompressor.codeplex.com/>。

另外 Google 也推出了自己的压缩工具 Closure Compiler，网址是 <http://code.google.com/closure/compiler/>。

对于多个 js 文件，可以将其合并成一个文件后，再进行压缩，比如可以将 Core.js、所有的公共控件、jQuery 类库压缩到一个 js 文件中，加上页面自身的 js 文件，达到上一节期望的一个页面只引用两个 js 文件的效果。

但是脚本压缩的劣势显而易见，查看压缩后的 js 文件，会发现 js 的源代码已经不具有可读性。一旦发现问题，将无法使用 firebug 等工具进行脚本调试。

另外一点，目前网站基本上都会启用 gzip 压缩，gzip 是在网络传输中的一种数据压缩，对于 js 文件压缩效率非常高。对 js 文件进行文件压缩后再进行 gzip 压缩，与不压缩 js 文件只进行 gzip 压缩相比，虽然也能减少传输，但是效果的提升不是很明显。

所以在是否对 js 文件进行文件压缩，需要项目设计人员根据自身需求做权衡。如果项目的 js 文件不是十分庞大，或者应用于内部网络不需要考虑网络问题，则可以不适用脚本压缩，只启用 gzip 压缩即可。



## 13.2 公共脚本类库

除了上面提到的 `Class.create`，还有很多类似的公共方法都需要放到 `Core.js` 文件中。

利用 JavaScript 的原型机制，可以为 JavaScript 的原生类型添加方法。在本节将介绍几个在接下来的例子中必须要用到的公共类库方法。

### 13.2.1 template 模板方法

首先介绍的就是 `string` 类型的模板方法：

【代码路径：jQueryStorm.Web/static/common/js/Core.js】

```
jQuery.extend(String.prototype, {
    template: function(object)
    {
        var regex = /#{(.*)}/g;
        return this.replace(regex, function(match, subMatch, index, source){
            return object[subMatch] || "";
        });
    }
});
```

通过扩展 `String.prototype`，为 `string` 类型的对象添加了 `template()` 方法，此方法能够用传入的对象属性替换字符串中的属性占位符。比如：

```
"my name is #{name}".template({name:"ZhangZiqiu"}); //output:my name is ZhangZiqiu
```

占位符的规则是“#{属性名称}”，上面的例子中会将传入对象的 `name` 属性值替换占位符的内容并返回。

### 13.2.2 修改函数上下文的方法

在讲解 `jQuery.proxy` 时曾讲过，在用户事件触发时，比如单击事件，事件处理函数中的 `this` 是指向页面对象的。虽然 `jQuery.proxy` 能够替换 `this`，但是语法不够优美。

所以，我们在 `Core.js` 中稍加修改，为 `function` 类型的对象添加一个 `proxy()` 方法。

【代码路径：jQueryStorm.Web/static/common/js/Core.js】

```
jQuery.extend(Function.prototype, {
    proxy: function (context) {
        return jQuery.proxy(this, context);
    }
});
```

同使用 `jQuery.proxy()` 方法进行对比，会发现使用 `jQuery.proxy()` 方法更方便、语法更优雅：

```
//使用 jQuery.proxy() 方法
this.$checkInTime.click( jQuery.proxy(this.onCheckInTimeClick));

//使用 bind() 方法
this.$checkInTime.click(this.onCheckInTimeClick.proxy(this));
```

在页面脚本对象中，为对象绑定事件的时候一定要使用 `proxy()` 函数，这是因为在页面初始化的时候，各种用户事件并没有触发。单击按钮触发绑定的事件时，事件处理函数内



部的 this 已经不是页面脚本对象本身了，也就无法使用 this 引用页面脚本对象的各个属性。而在绑定事件时，通过上面的方式修改 this 对象，可以解决此问题。

实际上实现这种功能是利用了 JavaScript 中的闭包。有关闭包的介绍可以参考“必须知道的 JavaScript”一章的内容。

### 13.2.3 反序列化 unparam()方法

jQuery 提供了 AJAX 帮助函数 jQuery.param(), 在“使用 AJAX 增加用户体验”一章对这个函数有详细的讲解。

jQuery.param()可以将对象序列化成参数字符串，而这个字符串也恰好可以用做 Cookie 值，所以下面操作 Cookie 的函数实现上需要用到 jQuery.param()方法。不幸的是 jQuery 并没有提供一个反序列化的方法，即将字符串反序列化成对象。

下面是 Core.js 中 unparam 方法的实现：

【代码路径：jQueryStorm.Web/static/ common/js/Core.js】

```
jQuery.unparam = function (value)
{
    var params = {};
    var pairs = value.split('&');
    if (value.indexOf("=") < 0)
    {
        return value;
    }

    for (var i = 0; i < pairs.length; i++)
    {
        var pair = pairs[i].split('=');
        var accessors = [];
        var name = decodeURIComponent(pair[0]), value = decodeURIComponent(pair[1]);

        var name = name.replace(/\[([^\]]*)\]/g, function (k, acc) { accessors.push(acc);
        return ""; });
        accessors.unshift(name);
        var o = params;

        for (var j = 0; j < accessors.length - 1; j++)
        {
            var acc = accessors[j];
            var nextAcc = accessors[j + 1];
            if (!o[acc])
            {
                if ((nextAcc == "") || (/^[0-9]+$/.test(nextAcc)))
                    o[acc] = [];
                else
                    o[acc] = {};
            }
            o = o[acc];
        }
        acc = accessors[accessors.length - 1];
        if (acc == "")
            o.push(value);
        else
            o[acc] = value;
        }
    return params;
};
```







unparam()函数就是对 param()的反向序列化。其中主要是部分算法函数。下面主要讲解其使用。

对于下面的对象:

```
var user = { name:"子秋", age:99 }
```

使用 param()方法, 进行序列化:

```
var userString = $.param(user); // 输出:"name=%E5%AD%90%E7%A7%8B&age=99"
```

现在可以使用 unparam()方法, 将 userString 反序列化成对象:

```
$.unparam(userString );
```

### 13.2.4 操作 Cookie 的方法

有时需要使用脚本操作用户的 Cookie。通常在服务器端也需要对 Cookie 进行操作, 所以在编写操作 Cookie 的脚本时, 需要考虑与服务器端的兼容性问题。

Cookies 是十分宝贵的客户端资源, 大多数浏览器支持最大为 4096 字节的 Cookie。浏览器还限制站点可以在用户计算机上存储的 Cookie 的数量。大多数浏览器只允许每个站点存储 20 个 Cookie; 注意这里的 20 个是指主键值, 也就是 20 条 Cookies 记录, 但是每个 Cookies 记录还可以包含若干子键, 如果试图存储更多 Cookie, 则最旧的 Cookie 便会被丢弃。有些浏览器还会对它们将接受的来自所有站点的 Cookie 总数作出绝对限制, 通常为 300 个。有关 Cookie 的详细信息, 可以参见相关的文章, 网址为:

<http://www.cnblogs.com/zhangziqui/archive/2009/08/06/cookies-javascript-aspnet.html>。

正因为 Cookie 资源的宝贵, 所以使用 Cookie 子键是十分重要的。在本文中提供的操作 Cookie 的方法就支持对 Cookie 子键操作。

为 jQuery 添加 getCookie 和 setCookie 两个工具函数, 源代码如下:

【代码路径: jQueryStorm.Web/static/ common/js/Core.js】

```
jQuery.getCookie = function (name, subName)
{
```

```
    // <summary>获取 Cookies 信息, 根据 Cookies 内容返回对象会单值。不支持多层次嵌套的对象结构。
```

使用举例:

```
    // 获取主键为 user 的 cookie: $.getCookie("user"); (如果 user 含有子键, 则返回对象)
```

```
    // 获取主键为 user, 子键为 name 的 cookie: $.getCookie("user", "name");
```

```
    // </summary>
```

```
    // <param name="name" type="string">cookie 主键</param>
```

```
    // <param name="subName" type="string">cookie 子键, 可省略</param>
```

```
    // <returns type="object">cookie 值, string 或 object 类型</returns>
```

```
    var cookieValue = "";
```

```
    if (document.cookie && document.cookie != "")
```

```
    {
```

```
        var cookies = document.cookie.split(';');
```

```
        for (var i = 0; i < cookies.length; i++)
```

```
        {
```

```
            var cookie = jQuery.trim(cookies[i]);
```

```
            if (cookie.substring(0, name.length + 1) == (name + '='))
```

```
            { //获取匹配主键名称的 cookie 字符串
```

```
                cookieValue = decodeURIComponent(cookie.substring(name.length + 1)); //获取 Cookies
```

```
                cookieValue = $.unparam(cookieValue);
```

主键值





```

        if (arguments.length > 1 && typeof arguments[1] === "string")
        { //获取 Cookies 子键值
            if (typeof subName !== 'undefined' && subName !== null && subName !== "")
            {
                cookieValue = cookieValue[subName] || "";
            }
        }
        break;
    }
}
}
return cookieValue;
}

```

```

jQuery.setCookie = function (name, value, options)
{
    /// <summary>设置 Cookies 信息。value 不支持多层次嵌套的对象结构。</summary>
    /// <param name="name" type="string">cookie 主键</param>
    /// <param name="value" type="string">cookie 值，可以是对象。</param>
    /// <param name="options" type="object">设置参数 map 对象，包括以下属性：
    /// expires[过期时间，数字(秒)或 Date 对象]
    /// path[路径，默认为根目录]
    /// domain[作用域，string，默认为根域名]
    /// secure[是否加密，传递任意值均表示加密]
    /// rewrite[是否覆盖，如果 value 传递对象则默认会和 cookie 的值做合并，将 rewrite 设置为 true 则可以
清空 cookie 并写入新值]
    /// </param>
    options = options || {};
    if (value === null)
    {
        value = "";
        options.expires = -1;
    }
    var expires = "";
    if (options.expires && (typeof options.expires === 'number' || options.expires.toUTCString))
    {
        var date;
        if (typeof options.expires === 'number')
        {
            date = new Date();
            date.setTime(date.getTime() + (options.expires * 24 * 60 * 60 * 1000));
        } else
        {
            date = options.expires;
        }
        expires = '; expires=' + date.toUTCString(); // use expires attribute, max-age is not supported by IE
    }
    var path = options.path ? '; path=' + (options.path) : ';path=/';
    var domain = options.domain ? '; domain=' + (options.domain) : ";";
    var secure = options.secure ? '; secure' : "";

    if (typeof value === "object")
    {
        if (options && !options.rewrite)
        { //rewrite 为 true 表示重写此 Cookie 值，不做合并处理
            var currentValueObj = $.getCookie(name, true); // 当前 Cookie 中的值，对象形式
            value = jQuery.extend(currentValueObj, value);
        }
        value = $.param(value);
    }
}

```



```
document.cookie = [name, '=', value, expires, path, domain, secure].join("");
}
```

上面操作 Cookie 的函数主要有两个，setCookie()和 getCookie()。基本满足了大部分 Cookies 操作的需求。下面是主要应用场景。最简单的场景——添加一个主键的 Cookie。

```
$.setCookie("user", " ziqiu");
```

获取这个 Cookie 值。

```
$.getCookie("user");//输出:ziqiu
```

要使用子键，需要首先构造一个 Cookie 中要保存的对象，比如：

```
var user = { name:"子秋", age:99 }
```

然后，可以将其作为 Cookie 值保存。

```
$.setCookie("user", user);
```

setCookie 会自动识别传入的 user 对象，并且将 user 的每一个属性作为一个子键保存。在获取的时候有几种方式，可以通过某一个具体的子键值：

```
$.getCookie("user", "age");//输出： 99
```

如果只获取主键值，则会返回一个对象，可以通过对象的属性获取子键值。

```
$.getCookie("user").age; //等价于$.getCookie("user", "age")
```

setCookie()函数还有几个特性。首先如果传递的是对象，比如一个只有 name 属性的对象，而此时 Cookie 已经存在，并且是一个只有 age 属性的对象，则 setCookie 会将此条 Cookie 合并，即保存后 Cookie 对象将包括 age 和 name 两个属性。实际上此功能主要是为了单独修改某一个子键准备的。修改一个子键值而不影响其他的子键，所以要做合并处理。下面的例子说明了这一特性。

```
var u1 = { name: "ziqiu" };
var u2 = { age:99 };
$.setCookie("user",u1);
$.setCookie("user",u2);
$.getCookie("user");// 输出: {name:"ziqiu", age:"99"}
var u3 = { age:18 };
$.setCookie("user",u3);
$.getCookie("user");// 输出: {name:"ziqiu", age:"18"}
```

另外一个重要参数就是 setCookie 方法的可选参数“options”，包括下面几个属性值。

- ☐ expires: 过期时间，数字（秒）或 Date 对象。
- ☐ path: 路径，默认为“/”。
- ☐ domain: Cookie 作用域，string 值，默认为当前域名。
- ☐ secure: 是否加密，传递任意值均表示加密。
- ☐ rewrite: 是否覆盖，如果 value 传递对象则默认会和 cookie 的值做合并，将 rewrite 设置为 true 则可以清空 cookie 并写入新值。



### 13.2.5 JSON 转换方法

在 AJAX 一章，曾经介绍了一个工具函数，用来作为 JSON 的转化。Core.js 中就包括此函数，具体使用请参见“使用 AJAX 增加用户体验”章节的相关内容。

Core.js 中不仅仅只有这些方法。只要是公共的通用方法，都可以放到这个公共类库文件中。

## 13.3 打造 jQuery UI 控件库

通常页面上会有很多的公共控件，比如日历、城市选择框、弹出层等。将这些公共的 UI 封装成控件是每个网站都会做的工作。下面介绍几种常用的打造网站 UI 控件库的方式。

### 13.3.1 使用 jQuery UI

在 jQuery UI 一章中已经详细介绍了 jQuery UI 的各个控件。使用 jQuery UI 可以不用自己编写核心代码，直接使用 jQuery UI 提供的各种菜单、日历、弹出层、自动建议等控件。

下面将使用 jQuery UI 的 Autocomplete 和 datpicker 插件，打造一个艺龙旅行网的机票搜索框。艺龙旅行网拥有全球最大的网站联盟，用本实例介绍的方法，任何人都可以申请一个艺龙联盟网站，然后在自己的网站上使用本实例打造的搜索框搜索机票。虽然有广告嫌疑，但是不可否认加入艺龙联盟是将网站流量转换成现金收益最好的方法之一。

首先先来看一下具体实现后的效果。如图 13-4 所示为使用 jQuery UI 的 Autocomplete 控件的效果。

如图 13-5 所示为使用 jQuery UI 的日历控件的效果。

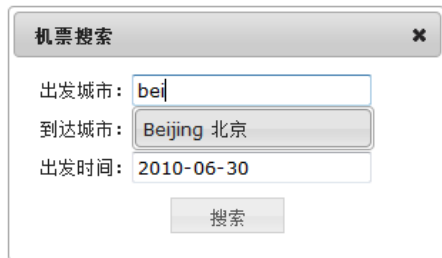


图 13-4 Autocomplete 控件效果



图 13-5 使用日历控件效果

在打造本实例的过程中，有以下几个关键点。



## 1. 加载日历控件的中文语言包

日历控件默认是英文，要改成中文需要将日历控件中文包的内容复制到 Core.js 中，代码如下：

```
jQuery(function($){
    $.datepicker.regional['zh-CN'] = {
        closeText: '关闭',
        prevText: '<#x3c;上月',
        nextText: '下月>#x3e;',
        currentText: '今天',
        monthNames: ['一月','二月','三月','四月','五月','六月',
            '七月','八月','九月','十月','十一月','十二月'],
        monthNamesShort: ['一','二','三','四','五','六',
            '七','八','九','十','十一','十二'],
        dayNames: ['星期日','星期一','星期二','星期三','星期四','星期五','星期六'],
        dayNamesShort: ['周日','周一','周二','周三','周四','周五','周六'],
        dayNamesMin: ['日','一','二','三','四','五','六'],
        weekHeader: '周',
        dateFormat: 'yy-mm-dd',
        firstDay: 1,
        isRTL: false,
        showMonthAfterYear: true,
        yearSuffix: '年';
    };
    $.datepicker.setDefaults($.datepicker.regional['zh-CN']);
});
```

语言包主要提供了月份、星期、数字等日历框相关的中文文字。在插件使用时会依赖这些文字。

## 2. 使用 Autocomplete 控件

自动建议控件的数据源保存在了页面脚本对象的 \$citys 属性中，在 initializeDOM 事件中赋值：

```
this.$citys = [{ label: "Beijing 北京", value: "北京", cityId: "BJS" }, { label: "Shanghai 上海", value: "上海", cityId: "SHA" }];
```

本实例的数据非常简单，只列举了两个城市，北京和上海。其中 label 属性自动建议控件用于检索的属性，value 属性是控件用来填充 input 的值，而 cityId 是一个隐藏属性，用户是不会看到的，这是城市的三字码，在机票系统中搜索实际上是使用城市三字码的。当用户选中了一个城市时，会在隐藏域中填充此属性。实现此行为是通过重写自动建议控件的 select() 函数实现的，代码如下：

```
this.$startCity.autocomplete({
    source: this.$citys,
    select: function (event, ui) {
        jQuery("#startCity").val(ui.item.value);
        jQuery("#hStartCity").val(ui.item.cityId);
        jQuery("#endCity").click();
    },
    minLength: 0
});
```

上述代码在 select 事件中还做了一件事情，就是控制光标的跳转。这样用户选择第一个城市后，将立刻弹出第二个城市输入框，选择完第二个城市输入框后，会自动弹出日历选



择框。这是一种流畅的用户体验，可以提高用户的使用感受。

另外，希望当输入框为空并且被单击时，也能触发自动建议控件。所以为这两个城市输入框添加了两个单击事件，并在事件中手动触发城市输入框的 search 事件：

```
onStartCityFocus: function (e) {
    if ($(e.target).val() === "") {
        this.$startCity.autocomplete("search", "");
    }
},
onEndCityFocus: function (e) {
    if ($(e.target).val() === "") {
        this.$endCity.autocomplete("search", "");
    }
}
```

本实例是完全使用 jQuery 的皮肤样式和 jQuery UI 的控件打造的。下面是完整的页面脚本对象代码：

【代码路径：jQueryStorm.Web/Static/flight/js/FlightSearchBoxUI.js】

```
/// <reference path="jquery-1.4.2.js" />
/// <reference path="Core.js" />
var FlightSearchBoxUI = {

    initializeDOM: function () {
        this.$btnSearch = jQuery("#btnSearch");
        this.$divMsg = jQuery("#divMsg");
        this.$startCity = jQuery("#startCity");
        this.$endCity = jQuery("#endCity");
        this.$hStartCity = jQuery("#hStartCity");
        this.$hEndCity = jQuery("#hEndCity");
        this.$startTime = jQuery("#startTime");
        this.$citys = [{ label: "Beijing 北京", value: "北京", cityId: "BJS" }, { label: "Shanghai 上海", value: "上海", cityId: "SHA" }];
        this.$divFlightSearchBox = jQuery("#divFlightSearchBox");
    },
    initializeEvent: function () {
        this.$startCity.autocomplete({
            source: this.$citys,
            select: function (event, ui) {
                jQuery("#startCity").val(ui.item.value);
                jQuery("#hStartCity").val(ui.item.cityId);
                jQuery("#endCity").click();
            },
            minLength: 0
        });

        this.$endCity.autocomplete({
            source: this.$citys,
            select: function (event, ui) {
                jQuery("#endCity").val(ui.item.value);
                jQuery("#hEndCity").val(ui.item.cityId);
                jQuery("#startTime").focus();
            },
            minLength: 0
        });

        this.$startTime.datepicker({ dateFormat: 'yy-mm-dd', minDate: new Date() });
        this.$startCity.click(jQuery.proxy(this.onStartCityFocus, this));
        this.$endCity.click(jQuery.proxy(this.onEndCityFocus, this));
        this.$btnSearch.click(jQuery.proxy(this.onBthSearchClick, this));
        this.$divFlightSearchBox.dialog({ position: [30, 30], resizable: false });
    },
}
```



```

pageLoad: function () {
},
onBthSearchClick: function () {
    if (!this.$startCity.val()) {
        alert("请选择出发城市！");
        return false;
    }

    if (!this.$endCity.val()) {
        alert("请选择到达城市！");
        return false;
    }

    if (!this.$startTime.val()) {
        alert("请选择出发时间！");
        return false;
    }

    var url = "http://" + "travel.elong.com/flight/cn_list_" + this.$startCity.val() + "_" + this.$endCity.val()
+ "_" + this.$startTime.val() + "_Y.html?campaign_id=4055103";
    location.href = url;
},
onStartCityFocus: function (e) {
    if ($(e.target).val() === "") {
        this.$startCity.autocomplete("search", "");
    }
},
onEndCityFocus: function (e) {
    if ($(e.target).val() === "") {
        this.$endCity.autocomplete("search", "");
    }
}
}

$(function () {
    var flightSearchBoxUI = Class.create(FlightSearchBoxUI);
})

```

完整的页面文件代码，可以在本书代码中的下列路径找到：

【代码路径：jQueryStorm.Web/chapter13/FlightSearchBoxUI.aspx】

使用 jQuery UI 可以用最少的工作实现很丰富的用户效果，但是也有一些不足。比如对于控件的控制不足，一些 bug，比如切换控件时无法控制焦点，功能不足，比如 Autocomplete 只能搜索 label 字段等问题都无法快速地解决。而且虽然控件功能强大烦琐，但是往往用不到那么多的配置参数。另外 jQuery UI 库也很大，如果将 UI 库也放到公共类库 core.js 中，会导致此文件过大。

所以在一些开发能力较强的公司，往往使用自己制作的精简的 UI 控件库。下面以日历控件为例，提供给大家一个最佳实践。

### 13.3.2 自定义日历控件

日历是任何网站都要使用的控件，虽然 jQuery UI 的 Datapicker 日历控件提供了基本的功能和各种扩展，但是常常依然不能满足复杂的业务需求，并且 jQuery UI 只选择日历控件的类库 js 文件大小也有几十兆，最后此自定义日历控件只有 200 多行代码。

本实例将重点讲解如何制作一个控件，从设想、设计到实现的整个过程。讲解设计方



法而不是像 jQuery UI 一样只讲解如何使用，授人于“渔”远胜于授人于“鱼”，如果没有自己的作品，永远都无法得到真正的飞跃。

### 1. 分析功能需求

当设计一个 UI 控件时，首先需要列出控件需要支持的功能列表：

- ☐ 双日历，支持向前、向后调整月份。
- ☐ 支持选择某一个日期。
- ☐ 支持设置日期默认值。
- ☐ 支持设置可选日期范围。
- ☐ 多个日历控件支持联动，比如选择“入住日期”后自动弹出“离店日期”日历框。
- ☐ 支持“yyyy-mm-dd”（中文）和“mm/dd/yyyy”（英文）两种格式。

由此看出日历框的功能还是比较简单的。

### 2. 设计对象模型

有了一个最终功能设想，现在就是要使用面向对象的方式设计日历控件。

首先，将日历控件定义为一个类，名字叫做 TopCalendar：

```
var TopCalendar = {
}
```

接着要设想这个类需要的属性和方法。日历框最主要的方法就是显示、隐藏和选中日期。对于日历的显示和隐藏通常有两种做法，一种是将日历控件添加到页面上，控制日历控件的 display 属性。另一种是每次显示时生成新的日历控件对象，隐藏时即销毁。

本实例使用第二种做法，即在日历控件“加载”的时候，即显示控件，可以在 pageLoad 事件做这件事。控件的销毁则需要设计一个 dispose() 方法。加上日期选中、月份调整这两个事件及前面提到的创建一个类时一个类的各种初始化事件，就已经有了一个大概的对象设计了，代码如下：

```
var TopCalendar = {
    initialize: function() {},           //构造函数
    initializeDOM: function() {},        //加载 DOM
    initializeEvent: function() {},      //绑定事件
    pageLoad: function() {},            //控件显示
    dispose: function() {},              //控件掩藏（同时销毁控件占用的资源）
    onRegionClick: function() {}        //单击日历控件的事件，将“选择日期”和“更改月份”合并到同一个事件中
}
```

很多 jQuery 控件会将各种用户事件，比如单击按钮等也放在控件中绑定。但是从面向对象的角度，这个职责其实是错误的。这样做让用户事件与控件形成了耦合，而且不利于控件的扩展。绑定用户事件应该在页面事件中完成。划分职责也会让设计人员思路更清晰。所以在这里 TopCalendar 中除了对日历框自己的 DOM 元素的事件处理之外，不包含和页面其他元素的事件绑定。

对于日历控件类，还需要定义一些属性，首先就是和控件类相关的 DOM 元素，代码如下：

```
// DOM
$contentEndRegion: null, //控件公共容器
```







```
$windowElement: null,    //日历主窗口对象
$eventElement: null,     //绑定日历的对象
```

\$windowElement 就是日历弹出层, \$eventElement:是绑定日历的控件对象, 一般是 input 对象。\$ contentEndRegion 是需要说明的, 这是一个容器 DOM, 所有的类似于日历控件的各种弹出层 DOM, 都需要放在此元素中, 这样才可以更好地组织页面。所以在页面 HTML 中, 会加入这样一个 div。

```
<div id="m_contentend">
</div>
```

所有的属性都可以通过构造函数的参数进行传递, 代码如下:

```
initialize: function (options) {
    jQuery.extend(this, options);
    var inputDate = this.getDateByString(this.selectDateString);
    this.year = inputDate.year;
    this.month = inputDate.month;
    this.day = inputDate.day;
}
```

还记得 jQuery.extend()函数的作用吗? 如果忘记了可以回头查看“jQuery 工具函数”一章。使用 jQuery.extend()函数, 将传递过来的 options 对象与日历框对象进行合并。也就是说, 如果外部传进一个\$ eventElement 对象, 则日历框控件内的\$ eventElement 也会被赋值, 比如下面是使用日历框的一个例子:

```
onCheckInTimeClick: function (event) {
    if (!(this.checkInCalendar && this.checkInCalendar.$windowElement)) {
        this.checkInCalendar = Class.create(TopCalendar, {
            $eventElement: this.$checkInTime,
            onSelected: function () {
                jQuery("#checkOutTime").trigger("click");
            }
        });
    }
    event.stopPropagation();
}
```

在上面的例子中, 传入了 onSelected 属性, 类型是函数, 会在用户选中日期后触发。在 TopCalendar 类中本身就预留了 onSelected 属性, 只要在初始化时传入了此属性, 则 TopCalendar 类自身的 OnSelected 属性就会被覆盖。在 TopCalendar 类的涉及内部, 可以通过 this.OnSelected 访问到此属性。

因为日历框控件的需求中还有多语言、可设置可选日期等功能, 所以还为对象添加了相应的属性。这些属性可以在开发过程中再补充完善。

UI 控件对象设计的思想精髓就是只考虑控件对象自身的行为, 而不去考虑如何与 DOM 对象做关联。

### 3. 设计用户交互

日历控件对象现在已经有了现实和隐藏的方法, 接下来就要考虑如何让日历控件与用户交互。

日历控件通常是应用在一个 text 类型的 input 输入框中, 通常具有如下行为:

❑ 当用户单击输入框时弹出日历控件。





- ❑ 单击日历框控件选择日期，可选：自动跳转并弹出另一个日历框控件。
- ❑ 单击日历框控件切换月份。
- ❑ 单击日历框控件关闭按钮。
- ❑ 单击页面其他地方，日历框自动关闭。

这是日历控件的常用交互行为，但是也可能存在特殊的使用场景，比如单击一个图片，弹出一个日历框，在选择某一个日期时，不是简单地输入框中填充日历，而是触发另一个和日期相关的复杂应用。这就是为什么在这里一再强调的，将控件的自身行为与页面交互（页面作为控件的消费者）相分离。在页面脚本对象中，需要编写控制日历控件的事件，以一个输入框空间为例，要为 input 绑定单击事件：

```
onCheckInTimeClick: function (event) {
    if (!(this.checkInCalendar && this.checkInCalendar.$windowElement)) {
        this.checkInCalendar = Class.create(TopCalendar, {
            $eventElement: this.$checkInTime,
            onSelected: function () {
                jQuery("#checkOutTime").trigger("click");
            }
        });
    }
    event.stopPropagation();
}
```

使用 Class.create() 函数，创建一个 TopCalendar 实例，会触发控件的 pageLoad() 函数，显示一个日历框对象。日历框对象保存在页面脚本类的 “this.checkInCalendar” 属性中。在生成日历框的时候，传入了 onSelected 属性，可以实现当选中了酒店的“入住日期”时，自动弹出“离店日期”日历框。

在失去焦点的时候隐藏日历框，是通过添加失去焦点事件实现的：

```
onCheckInTimeBlur: function (event) {
    if (this.checkInCalendar && this.checkInCalendar.$windowElement) {
        this.checkInCalendar.dispose();
        this.checkInCalendar = null;
    }
}
```

this.checkInCalendar 属性是日历框对象，但是日历框对象不仅仅是一个 DOM，更是一个包含着很多属性和函数的内存实例。其中它的一个属性 \$windowElement 就是页面上日历窗口对象，如果日历对象已经执行了 dispose() 方法，则此属性将为 null。这里用于判断对象为 null 的方法，使用了 JavaScript 中特有的方式。在“必须知道的 JavaScript 知识”一章中讲过 “&&” 运算符，如果 this.checkInCalendar 属性为 null，在 “&&” 运算中会直接返回 false，否则会继续判断 this.checkInCalendar.\$windowElement 是否为 null。“&&” 和 “||” 的使用能够使 JavaScript 代码看起来更优雅。

至于日历框自身的各种事件，是包含在日历的对象模型中的。一个页面只需要编写上面这两个事件，就可以使用日历控件了。

#### 4. 日历框的内部事件

在日历框内部，有以下三个事件：

- ❑ 更换月份。





❑ 选择日期。

❑ 关闭窗口。

传统的事件绑定，是为不同的 DOM 对象绑定不同事件处理函数。但是这里介绍另外一种事件处理的方式，为日历框的最外层 div 绑定一个单击事件，在单击事件处理函数内部时，根据触发事件的 DOM 不同，执行不同的处理逻辑。下面是 TopCalendar 的弹出层的几个 HTML 片段：

按钮的 HTML 代码片段。

```
<a class="mf_lr_a" title="上一月" href="#" method="btnPre">
<a class="mf_rr_a" href="#" title="下一月" method="btnNext">
```

关闭按钮的 HTML 代码片段。

```
<a href="#" title="关闭" class="ac_close_t cur" method="close">x</a>
```

某一个日期的 HTML 代码片段。

```
<td method="btnDay" class="selected" onmouseout="$(this).toggleClass('hover')"
onmouseover="$(this).toggleClass('hover')">5</td>
```

注意到在页面元素的 HTML 属性中，有“method”这一个自定义属性。在单击事件中正是根据这个属性来判断的，下面是单击事件的代码结构。

```
onRegionClick: function (event) {
    var elem = $(event.target);
    var method = elem.attr("method");
    switch (method) {
        case "btnPre"://上一个月
            .....
            return;
        case "btnNext"://下一个月
            .....
            return;
        case "btnDay"://选择某一天
            .....
            return;
        case "close"://关闭控件
            this.dispose();
            return;
    }
}
```

注意在 switch 分支中，是如何使用 method 属性来判断事件的。

接下来只需要在日历控件初始化事件的时候，为日历控件的 DOM 对象绑定此事件即可：

```
this.$windowElement.bind("mousedown", this.onRegionClick.bind(this));
```

需要特别注意的是，这里绑定的事件类型是 mousedown 而不是 click，这是因为如果选中日历控件的某一天，会触发页面上 input 元素的 blur 失去焦点事件，并且 blur 先于 click 触发。在上面的用户交互中，已经为 input 元素的 blur 事件设置为隐藏日历框，所以如果 blur 先触发则永远无法选中日期。解决办法就是使用 mousedown 事件。mousedown 事件会先于 blur 事件触发。

现在，日历控件的结构已经十分清晰了。补全上面的日历等各种内部函数，一个完整



的日历输入框就完成了。先来看一下最后实现的效果，如图 13-6 所示。

城市:

入住日期:

离店日期:

价格范围:

酒店名称:

2010年7月 2010年8月

日	一	二	三	四	五	六
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

日	一	二	三	四	五	六
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

图 13-6 实现效果

有关日历控件的完整代码，可以在下面的路径中找到：

【代码路径：jQueryStorm.Web/static/ common/js/TopCalendar.js】日历控件使用到的 CSS 作为网站基础的样式放在了 common.css 中，在 CSS 中也是使用相对路径引用的图片，所以可以将 CSS 和图片这种静态文件随意地迁移。

日历控件仅仅是网站众多控件中的一小部分，需要学习的不是一个控件的使用，而是如何设计与开发实现一个控件。只有掌握了方法，才能创造出属于自己的作品。

## 13.4 小结

本章分享了以 jQuery 为基础，一个网站脚本框架的概貌，从页面的脚本对象创建，到脚本文件的管理、压缩，以及如何创建和使用脚本框架，这是一种使用 jQuery 的最佳实践。其中也有需要设计人员根据项目需要来决定的事情，比如是否需要脚本压缩、使用 jQuery UI 的控件还是自己创建控件等。

希望读者在阅读学习本章的时候，更注重学习其中的方法，因为这些与使用何种类库无关，对于其他的脚本类库也同样适用。





## 第 14 章



# jQuery 与百度地图实战

本章将通过一个网站的策划、设计与实现，来讲解 jQuery 在实际产出中的应用。本章实例将使用 jQuery 配合百度地图，构建一个地图查询网站。

说到“地图”大家都不会陌生，它与目前最火爆的“我 LBS”应用密切相关。所谓“LBS”即基于位置的服务（Location Based Service, LBS），即通过通信设备获取到用户的位置信息，根据用户位置开展的一系列应用。美国的 LBS 应用网站 Foursquare 的发展势头和前景被认为超过当年的 Twitter（微博业务）和 Groupon（团购业务）。而在 LBS 应用中，地图是不可或缺的角色。所以本实例将构建一个基础地图网站。

本章实例只是一个简单的举例，其中比如网站规划等阶段并没有深入地研究，只是希望通过此网站实例让开发人员了解网站完整的产出过程。在百度地图 API 的帮助下，本实例只使用 jQuery 搭建，没有使用任何的服务器端程序。

## 14.1 网站规划

在开始制作网站前，首先会对网站进行一些规划。比如网站的主题、用户人群、盈利模式、未来发展等。本实例并没有深入地思考所有的问题，但是如果是一个准备长期投入的项目，就一定会在项目开始前对每一点进行深入思考和分析。

### 14.1.1 网站主题

网站主题是网站的灵魂，任何网站都有一个主题。比如游戏网站、音乐网站、门户网站等等。网站的主题通常决定了网站今后的走向，当然一个网站的主题定位也有可能产生变动。比如网易是社区起家，但是现在的主要收入是网游。网站主题也往往和盈利模式相关联。比如社交类网站的主要收入可能是广告，而电子商务类网站的主要收入是用户订单。

本实例确定了网站的主题是提供免费地图服务。帮助人们查找地点以及公交、驾车路线。

### 14.1.2 用户人群

网站都会有主要的用户人群，虽然网站站长们都希望“通吃”所有用户，但是这是不理智也不现实的。网站主题一般决定了网站主要的细分人群。而网站设计时也要优先考虑这部分人群的偏好。比如大型游戏类网站的主要定位人群大部分是年轻人，所以网页的设计风格会尽量的“花哨”。而苹果公司的定位人群是时尚用户，所以网站的设计追求简洁美。

本实例因为是地图服务，所以定位人群比较广，是所有有地图需求的用户，但是以后本实例会推出“手机版”，让用户在电脑和手机上可以有一致的用户体验。所以手机地图用户也是本实例的定位人群。

### 14.1.3 盈利模式

网站的“价值”，取决于它的盈利模式。

一个网站，可以暂时没有盈利，但是一定要在创建初期就构想好可能的盈利模式。经历过互联网泡沫的人们都知道，没有明确盈利模式，只是“烧钱”的网站，终究难逃灰飞烟灭的命运。

目前大部分网站的主要盈利模式是广告，这并没有错。但是单纯依赖广告很难有所作为。LBS 之所以如此火爆，正是因为它有着非常独特的盈利模式：可以将用户与周围的商家连在一起。

### 14.1.4 未来规划

网站初期应该尽量有一个完整的设计，但是不一定实现所有的设计。可以有计划地分步实施，以便网站能尽早积累用户。

本实例目前是一个网页版的地图服务，但是以后会推出手机版——百度地图 API 支持 android 和 iphone 手机，这正是其得天独厚的优势。

上面的泛泛而谈，很简陋粗糙，只是想让本实例尽可能的涉及建设网站初期应该考虑



的事情。规划网站其实是一门大学问。

## 14.2 网站实现

对网站有了粗略的计划后，就要开始动手实施了。本实例只有一个页面，在页面上用户可以查找地点、查询公交和驾车路线。假设定义此页面名称为 BMap，下面就开始此页面的设计、开发工作。

有的时候有了想法，真正开始着手做的时候却不知从何下手，甚至有的人上来就是先实现页面功能，而到最后才发现实现的不是最初想要的。所以说，实现一个页面并不是最重要的，而一个页面的思考和实现过程才是最重要的。

### 14.2.1 定义页面结构

制作页面前，首先要确定页面的结构，即页面如何切分。首先大的框架，肯定是头、中、尾三大区域。于是，我们可以先用 div 写出页面的大框架：

```
<div id="header"></div>
<div id="content"></div>
<div id="footer"></div>
```

在内容部分，可以在左侧放地图，右侧放用户输入控件。于是定义 content 层的结构为：

```
<div id="content">
  <div id="mapContainer"></div>
  <div id="mapControl"></div>
</div>
```

在 mapContainer 中将放置地图，在 mapControl 中将放置用户输入的控件，以及查询的结果。比如查找“中关村”附近的“小吃”时，将所有的结果放在一个 div 中。所以我们还需要进一步对 mapControl 进行切分：

```
<div id="mapControl">
  <div id="tabs"></div>
  <div id="mapResults"></div>
</div>
```

其中 tabs 为用户输入控件的容器。起名为“tabs”，是因为本实例将使用 jQuery UI 的 tabs 控件。效果如图 14-1 所示。



图 14-1 用户输入区域

下面是此页面的完整页面结构：

```
<div id="header"></div>
<div id="content">
  <div id="mapContainer"></div>
```



```
<div id="mapControl" >
  <div id="tabs"></div>
  <div id="mapResults"></div>
</div>
</div>
<div id="footer"></div>
```

页面有了结构，接下来即可添加内容和样式。

## 14.2.2 实现样式

页面样式是使用 CSS 实现的，一般都会有专门的页面美工（用来制作页面的效果图片）和页面设计师（将图片切割成 HTML 和 CSS）来完成，然后才会交给开发人员。但是对于个人站长来说，他们是全能选手。

通过前面对 jQuery UI 的介绍，我们知道如今有了 jQuery UI 的 Theme “主题皮肤”，开发人员也能“设计出”漂亮的网站。本实例的美工部分就是使用 jQuery UI 的“Smoothness”主题实现的。

虽然大部分美化工作可以实现了，但是页面上元素位置的调整、间距的调整等还是要通过 CSS 实现。通常，在开始设计时，先在 HTML 元素上直接写 CSS，比如：

```
<div id="mapContainer" style="width: 600px; height: 550px; border: #ccc solid 1px;
float: left;">
</div>
```

使用内嵌样式可以快速调整页面样式。等到页面样式调整完毕后，再进行“页面重构”，将 CSS 进行分类和提取，最后的效果是将所有的内嵌样式抽象出来，比如：

```
<div id="mapContainer" class="mapFrame">
</div>
```

有了内容，有了样式。只是此时的内容是静态的。接下来就要通过脚本程序，实现页面的功能了。

在此，首先来看一下本实例的实现效果，如图 14-2 所示。

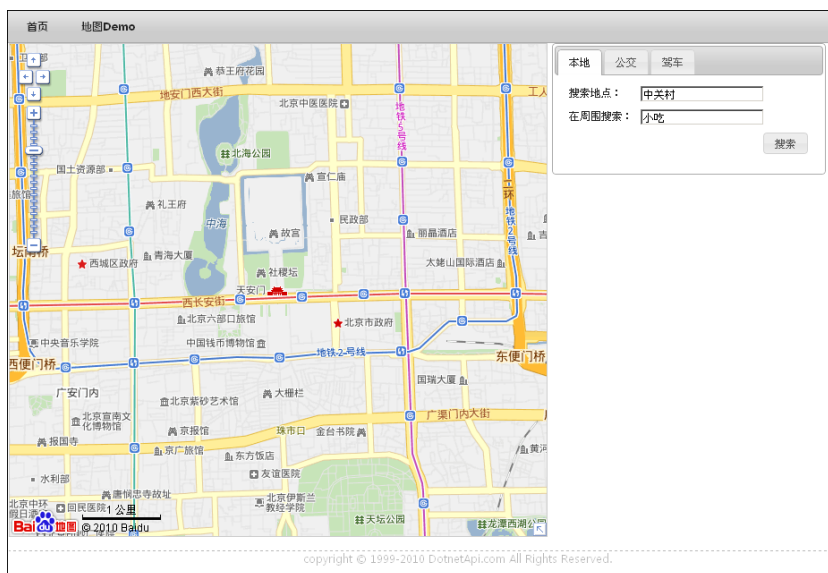


图 14-2 实例总体样式效果



### 14.2.3 实现页面功能

本实例全部的页面功能都是通过客户端脚本配合百度地图 API 接口实现的。在后面的小节中将单独讲解页面脚本。

### 14.2.4 页面重构

在实现页面功能的过程中，可能还会对页面样式进行微调。在功能实现完毕后，就需要对页面进行重构了。其中主要的工作就是将内嵌的 CSS 都提取出去。

本实例最终提取出来的 CSS 如下(省略内容，仅保留结构)：

```
/* 全局样式 */
*{...}
a{...}
a:hover{...}
body{...}
.clr{...}

/* 内容区域的样式 */
#content{...}
#mapContainer{...}
#mapControl{...}
#mapResults{...}

/* 用户输入 tabs 控件的样式 */
#tabs{...}
#tabs label{...}
#tabs .menuButton{...}
.menu div{...}

/* 页头的样式 */
#topMenu ul{...}
#topMenu ul li{...}
#headerBar {...}
#headerBar div{...}

/* 页尾的样式 */
#footerBar{...}
```

本实例的 CSS 样式大部分使用了 ID 选择器。因为页面结构上主要容器都有 ID。这样做的好处是不需要为容器添加额外的样式类，坏处就是复用性不够好，通常每一个容器的样式都是独立的。

最后再来看一下重构后的页面结构代码，已经不含有任何的 CSS 了：

```
<!-- 页头 -->
<div id="headerBar" class="ui-widget-header">
  <div id="logo"> DotnetApi.com </div>
  <div id="switchCity" ></div>
  <div id="topMenu">
    <ul>
      <li><a href="index.htm">首页</a></li>
      <li><a href="BMap.htm">地图 Demo</a></li>
    </ul>
  </div>
</div>
<!-- 内容 -->
<div id="content">
```





```
<!-- 百度地图容器-->
<div id="mapContainer"></div>
<!-- 用户控件即搜索结果容器 -->
<div id="mapControl"><!-- 内容略 --></div>
</div>
<div class="clr"></div>
<!-- 页脚 -->
<div id="footerBar">
    copyright © 1999-2010 DotnetApi.com All Rights Reserved.
</div>
```

本实例的完整代码，可以在下面路径中获取到：

【代码路径：jQueryStorm.Web/chapter14/BMap.htm】

至此，本实例的设计、实现过程就结束了。目前本实例只有一个页面，一个真实的网站往往会添加很多页面。但是都可以用相同的方法设计与实现。

接下来，将详细地讲解本实例的功能实现。

## 14.3 脚本详解

本实例使用 jQuery+百度地图 API 实现，所以首先要了解百度地图 API。

### 14.3.1 百度地图 API 介绍

百度推出自己的地图服务比较晚，但是因为本土化优势，所以百度地图有着很好的用户体验，速度很快。百度地图 API 是一套由 JavaScript 语言编写的应用程序接口，它能够帮助站长在网站中构建功能丰富、交互性强的地图应用程序。百度地图 API 不仅包含构建地图的基本接口，还提供了诸如本地搜索、路线规划等数据服务，可以根据自己的需要进行选择。地图 API 的服务是免费的，任何非盈利性网站均可使用。有关百度地图 API 的首页是：

<http://openapi.baidu.com/map/index.html>

首页如图 14-3 所示。



图 14-3 百度地图首页





在百度地图 API 的网站上，可以找到完整的实例参考以及 API 说明文档。

### 14.3.2 使用百度地图 API

使用百度地图 API，就如同使用一个个封装好了的脚本函数。要使用百度地图 API，首先需要在百度地图 API 的网站上，注册一个账户。点击首页的“注册百度地图 API”，按照提示即可注册一个账户，最后的成功页面会提供一个注册账户的 key 值，如图 14-4 所示。



图 14-4 成功创建百度地图 API 账号

我们需要牢记这个 key。当然如果丢失了可以来重新注册。页面只有使用有效的 key 才能够调用百度地图的各种服务。有一点需要注意，在注册账号时我们需要填写一个网址，比如 mytest.com。也就是说，这个 key 只有在 mytest.com 域名下的页面上使用。但是 localhost 这个域名是特殊的，此域名可以随意调用百度地图 API 接口，所以在开发过程中可以将本地的测试页面都设置成 localhost 这个域名。

接下来，在页面上引用上图 14-4 所示的脚本块：

```
<script type="text/javascript" src="http://api.map.baidu.com/api?
key=51e98f02065e0e922e0ca62228018206&v=1.1&services=true"></script>
```

如图 14-4 所示也提供了两个脚本块，它们的区别仅仅在于最后一个参数“services”。我们应该引用“services=true”。因为百度地图提供的如本地搜索、公交查询、驾车查询，都属于“服务”，即“services”的一部分。

完成了上面的工作后，页面就已经具有调用百度地图 API 的能力了。先通过一个简单例子进行测试：

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
  <title>Hello, World</title>
  <script type="text/javascript" src="http://api.map.baidu.com/api?
key=46ce9d0614bf7aefe0ba562f8cf87194&v=1.1&services=true">
```



```

</script>
</head>
<body>
  <div style="width: 520px; height: 340px; border: 1px solid gray" id="container">
  </div>
  <script type="text/javascript">
    var map = new BMap.Map("container");           // 创建 Map 实例
    var point = new BMap.Point(116.404, 39.915);    // 创建点坐标
    map.centerAndZoom(point, 15);                  // 初始化地图,设置中心点坐标和地图级别
  </script>
</body>
</html>

```

上面的例子十分简单，首先在页面上放一个 `div` 作为地图显示的容器，然后使用百度地图 API 提供的 `BMap` 命名下的 `Map` 类创建地图，关联容器，使用 `Point` 类创建一个点，然后将 `Map` 对象的中心设置成此 `Point` 点。

有关百度地图 API 提供各种类的详细说明，可以自行去百度地图 API 官网的“类参考”查询，如图 14-5 所示。



图 14-5 百度地图 API 参考手册

百度地图 API 的所有类都是在 `BMap` 命名空间下的。比如 `BMap.Map`、`BMap.Point` 等等。

### 14.3.3 使用页面脚本框架

在第 13 章中，我们讲解了如何组织及编写页面脚本。本实例也使用相同的面向对象的方式组织页面脚本。页面的脚本都封装在了 `ThisPage` 类下：

```

Var ThisPage = {
  Initialize: function(){...},
  initializeEvent: function(){...}
}

```

使用 jQuery 在页面的 DOM 结构加载完毕后创建实例：

```

$(function ()
{
  var thisPage = Class.create(ThisPage)
});

```



Class.create 会创建一个类实例，并且顺序调用 ThisPage 类的 Initialize 和 initializeEvent 初始化实例。

#### 14.3.4 使用 jQuery UI

本实例使用了 jQuery UI 的 Tabs 控件，首先按照 Tabs 控件的要求，组织 HTML 结构，下面以其中的一个菜单“本地”为例：

```
<div id="tabs">
  <!--ul 放置菜单项 -->
  <ul>
    <li><a href="#tabs-1">本地</a></li>
  </ul>
  <!--div 放置菜单项的内容，通过菜单项的 href 相关联 -->
  <div id="tabs-1" class="menu">
    <div>
      <label for="iptSearchFor">搜索地点： </label>
      <input id="txtSearchFor" name="searchFor" type="text" value="中关村" />
    </div>
    <div>
      <label for="iptSearchFor">在周围搜索： </label>
      <input id="txtSearchNearby" type="text" value="小吃" /><br />
    </div>
    <div class="menuButton">
      <input id="btnSearchFor" type="button" value="搜索" />
    </div>
  </div>
</div>
```

设置全部使用默认值，在 initialize 中初始化控件：

```
initialize: function ()
{
    /// <summary>
    /// 初始化
    /// </summary>
    $("#tabs").tabs();
    $("#btnSearchFor, #btnSearchTransit, #btnSearchDriving").button();
}
```

这里还使用了 button 控件，其实主要是用来为按钮添加一个好看的样式，并没有使用任何控件的功能。

通过上面的简单使用，就完成了了一个漂亮的用户输入控件栏，如图 14-6 所示。

图 14-6 用户输入控件栏

主要的功能都是在按钮单击时触发的。在 initializeEvent 事件中，我们为 3 个按钮绑定单击事件。

### 14.3.5 本地搜索

首先看本地搜索，下面是“本地”标签下，“搜索”按钮的单击事件的脚本代码：

```
$("#btnSearchFor").click(function (e)
{
    var searchFor = $("#txtSearchFor").val();
    var searchNearby = $("#txtSearchNearby").val();

    if (searchFor && !searchNearby)
    {
        this.local = new window.BMap.LocalSearch(this.map, {
            renderOptions: { map: this.map, panel: "mapResults" }
        });
        this.map.clearOverlays();
        this.local.search(searchFor);
    }
    else (searchFor && searchNearby)
    {
        this.local = new window.BMap.LocalSearch(this.map, {
            renderOptions: { map: this.map, autoViewport: false,
selectFirstResult: false },
            pageCapacity: 1
        });
        this.map.clearOverlays();
        this.local.setSearchCompleteCallback(this.onSearchComplete);
        this.local.search(searchFor);
    }
} .proxy(this));
```

本地搜索可以只输入一个地名，则功能是地点查找。比如输入“中关村”，则会在地图上搜索和“中关村”有关的地点。也可以同时填写“在周围搜索”，比如“小吃”，则会在地图上“中关村”附近搜索“小吃”有关的商家信息。

这些功能都是百度地图 API 的“本地搜索”数据服务提供的。“本地搜索”都封装在了“LocalSearch”类。下面的代码用于创建一个类实例：

```
this.local = new window.BMap.LocalSearch(this.map, {
    renderOptions: { map: this.map, panel: "mapResults" }
});
```

创建的时候，需要传递本地的 map 对象，即地图容器中的 map 对象。另外，panel 参数表示的是将搜索结果填充到哪个元素。

对于地点查找，直接使用 local 的 search 方法：

```
this.local.search(searchFor);
```

十分简单吧？但是别忘了，每次点击搜索，都会在地图上标注搜索结果，而且是累加的。所以在进行一次新的搜索前，需要清除上一次的搜索结果：

```
this.map.clearOverlays();
```

上面就是实现地点搜索的语句。如果填写了“在周围查找”，则要稍微复杂一点。其实百度地图 API 本身提供了一个简单的接口用来做“周边搜索”：

```
local.searchNearby("小吃", "前门");
```

但是 searchNearby 函数的结果不能让人满意，比如只能搜索出来最多 10 条结果，不能调节地图级别等。所以在本实例中，使用了功能更灵活更强大的“范围搜索”。配合“本地搜索”实现了上面的功能。



首先还是进行本地搜索，比如首先搜索“中关村”，但是添加了一个回调函数：

```
this.local.setSearchCompleteCallback(this.onSearchComplete.proxy(this));
```

在回调函数中，实现了我们自己实现的“周边搜索”：

```
onSearchComplete: function (results)
{
    /// <summary>
    /// 周边搜索的回调函数
    /// </summary>
    if (this.local.getStatus() == BMAP_STATUS_SUCCESS)
    { //搜索成功时，才执行
        var count = results.getCurrentNumPois();
        if (count > 0)
        {
            //首先搜索区域，获取第一个结果，并且将地图中心点转移到此地点
            var point = results.getPoi(0).point;
            this.map.centerAndZoom(point, 16);

            //在地图上，进行周边商户的搜索
            var local = new BMap.LocalSearch(this.map, {
                renderOptions: { map: this.map, selectFirstResult: false, panel: "mapResults" },
                pageCapacity: 20
            });
            this.map.clearOverlays();
            local.searchInBounds($("#txtSearchNearby").val(), this.map.getBounds());
        }
    }
}
```

此回调函数有一个参数 `results`，当此事件触发的时候，`results` 中会含有所有搜索“中关村”产生的结果。这些结果中的第一条通常是最准确的。所以首先获取第一条“中关村”地点的信息：

```
var point = results.getPoi(0).point;
```

然后，将地图的中心点设置为此 `point` 的位置，并且设置地图级别：

```
this.map.centerAndZoom(point, 16);
```

接着就可以在当前显示的范围，进行“范围搜索”了：

```
local.searchInBounds($("#txtSearchNearby").val(), this.map.getBounds());
```

下面来看一下该功能的效果。只搜索“中关村”，效果如图 14-7 所示。



图 14-7 本地搜索效果

在“中关村”周围搜索“小吃”，效果如图 14-8 所示。



图 14-8 周边搜索效果

### 14.3.6 公交和驾车搜索

相对于本地搜索，公交和驾车搜索就十分简单了，百度地图 API 本身提供的接口就能满足我们的需求。下面是公交和驾车的“搜索”按钮单击事件源代码：

```
$("#btnSearchTransit").click(function (e)
{
    ///<summary>
    ///公交搜索
    ///</summary>

    //获取用户输入
    var transitStart = $("#txtTransitStart").val();
    var transitEnd = $("#txtTransitEnd").val();

    if (transitStart && transitEnd)
    {
        //创建公交搜索 TransitRoute 对象，搜索路线
        var transit = new window.BMap.TransitRoute(this.map, {
            renderOptions: { map: this.map, panel: "mapResults", autoViewport: true }
        });
        this.map.clearOverlays();
        transit.search(transitStart, transitEnd);
    }
}.proxy(this));

$("#btnSearchDriving").click(function (e)
{
    ///<summary>
    ///驾车搜索
```





```
///</summary>

//获取用户输入
var drivingStart = $("#txtDrivingStart").val();
var drivingEnd = $("#txtDrivingEnd").val();

if (drivingStart && drivingEnd)
{
    //创建驾车搜索 DrivingRoute 对象，搜索路线
    var driving = new window.BMap.DrivingRoute(this.map, {
        renderOptions: { map: this.map, panel: "mapResults", autoViewport: true }
    });
    this.map.clearOverlays();
    driving.search(drivingStart, drivingEnd);
}
}.proxy(this));
```

和本地搜索十分相似，只是使用了不同的类实例。公交搜索使用“TransitRoute”类，通过“TransitRoute.search”方法进行搜索：

```
transit.search(transitStart, transitEnd);
```

而驾车搜索使用“DrivingRoute”类，通过“DrivingRoute.search”方法进行搜索：

```
driving.search(drivingStart, drivingEnd);
```

公交搜索的效果如图 14-9 所示。



本地 公交 驾车

起点:

终点:

搜索

起 北京站

1. 乘坐地铁2号线, 经过4站, 到达东直门, 步行约330米, 到达东直门, 乘坐地铁13号线, 经过10站, 到达西二旗 31.9 公里
2. 步行约200米, 到达北京站前街, 乘坐特2, 经过5站, 到达东直门北, 步行约220米, 到达东直门, 乘坐地铁13号线, 经过10站, 到达西二旗 32.5 公里
3. 乘坐地铁2号线, 经过8站, 到达积水潭, 步行约240米, 到达积水潭桥南, 乘坐88, 经过7站, 到达城铁大钟寺站, 步行约150米, 到达大钟寺, 乘坐地铁13号线, 经过4站, 到达西二旗 26.1 公里
4. 步行约1.1公里, 到达东单, 乘坐地铁5号线, 经过13站, 到达立水桥, 步行约120米, 到达立水桥, 乘坐地铁13号线, 经过4站, 到达西二旗 29.4 公里
5. 乘坐机场巴士3线, 经过2站, 到达东直门, 步行约140米, 到达东直门, 乘坐地铁13号线, 经过10站, 到达西二旗 32.3 公里

终 西二旗

[到百度地图查看»](#)

图 14-9 公交搜索效果

驾车搜索效果如图 14-10 所示。





图 14-10 驾车搜索效果

## 14.4 小结

本章通过百度地图 API 实例，从设想、设计到实现，完整地讲解了建设一个网站的流程。当然因为本实例过于简单，在这仅作抛砖引玉。但是从中我们可以发现百度地图 API 强大的功能，以及使用 jQuery 和 jQuery UI 高效的生产力。相信有了这些利器，网站开发人员可以发挥自己无穷的想象力，构建一个真正完美的网站！





## 第 15 章



# 移动脚本框架 jQuery Mobile

随着互联网各种应用的日趋饱和，移动互联网成为了众多商家下一个战场。随着 iPhone、Andriod 等手机的热卖，3G 网络的日益成熟，越来越多的用户开始了自己的无线生活。但是目前手机的应用还都是以“下载”——“安装”这种模式，也就是传统的 B/S 程序为主。随着移动互联网的发展，HTML5 的推广，越来越多的应用将会移植到浏览器端——就如同现在的互联网一样。

目前，开发手机浏览器应用最头疼的就是各种操作系统、各种浏览器的兼容问题。手机浏览器的种类比 IE 浏览器的还要多，而且差别特别大，比如 Window Mobile 手机自带的 IE 浏览器，几乎不能运行稍微复杂一些的脚本程序。好在 WM 手机的占用率很低。总之，想要实现一个跨手机浏览器的脚本框架，是一个异常艰难的任务。jQuery Mobile 就是为此目的而生的。虽然目前还是测试版本，但是已经能够兼容绝大部分手机浏览器。本章将对 jQuery Mobile 进行讲解，让 javascript 开发人员提前一步迈入移动互联网时代。

## 15.1 jQuery Mobile 介绍

在学习如何使用 jQuery Mobile 之前，先来认识一下什么是 jQuery Mobile。

### 15.1.1 jQuery Mobile 的目的

jQuery Mobile 官方将其定义为：应用在智能手机或平板电脑上，为触摸操作优化 Web 框架。以 jQuery 和 jQuery UI 为基础，能够跨越多种设备创建具有相同用户体验的系统。它的代码轻量、可持续改进、灵活性并且具有良好的模板设计。

也就是说，试用 jQuery Mobile，我们可以构建跨不同手机操作系统以及手机浏览器的 Web 应用。

注意这里所说的是 jQuery Mobile 的“目的”，目前还没有完全的实现，但是 jQuery Mobile 一直在向着这个目标努力着，并且已经实现了大部分的功能，已经能够支持 iOS、Andriod、黑莓、塞班等多种手机操作系统了。

### 15.1.2 jQuery Mobile 浏览器兼容性

因为手机操作系统和浏览器的复杂性，jQuery Mobile 即使是未来也不可能兼容所有的操作系统和浏览器。那么如何确定到底支持哪些操作系统和浏览器呢？jQuery Mobile 参考了 Yahoo 曾经使用过的方式，将手机浏览器和手机操作系统绘制一个表格，为组合进行打分。得分可能是 A、B、C 三种。jQuery Mobile 主要的工作是兼容所有得分是 A 的操作系统与浏览器组合。而对于得分是 B 和 C 的组合，将不会进行严格的测试，也不保证能够正常工作。

表 15-1 所示是手机浏览器得分表格。

表 15-1 手机浏览器得分表

Platform	Version	Native	Opera Mobile				Opera Mini		Fennec		Ozone	Netfront	Phonegap
			8.5	8.65	9.5	10.0	4.0	5.0	1.0	1.1	0.9	4.0	0.9
iOS	v2.2.1	A											A
	v3.1.3, v3.2	A						A					A
	v4.0	A						A					A
Symbian S60	v3.1, v3.2	C	C	C		B	C	B			C	C	
	v5.0	A	C	C		A	C	A					A
Symbian UIQ	v3.0, v3.1			C							C		
	v3.2				C						C		
Symbian Platform	3.0	A											
BlackBerry OS	v4.5	C					C	C					
	v4.6, v4.7	C					C	B					C
	v5.0	A					C	A					A
	v6.0	A						A					A
Android	v1.5, v1.6	A											A



(续表)

Platform	Version	Native	Opera Mobile				Opera Mini		Fennec		Ozone	Netfront	Phonegap
			8.5	8.65	9.5	10.0	4.0	5.0	1.0	1.1	0.9	4.0	0.9
Windows Mobile	v2.1	A											A
	v2.2	A				A		C		A			A
	v6.1	C	C	C	C	B	C	B				C	
	v6.5.1	C	C	C	C	A	C	A					
	v7.0	C				A	C	A					
webOS	1.4.1	A											A
bada	1.0	A											
Maemo	5.0	B				B			C	B			
MeeGo	1.1	A				A				A			

jQuery Mobile 只有在得分是 A 的浏览器上，才能获得很好的效果。因为这些浏览器的用户体验更好，功能更强。而用户也必然会越来越多，比如 iPhone 和 Andriod 系统自带的浏览器。

### 15.1.3 jQuery Mobile 特性

jQuery Mobile 具有如下的特性：

- ❑ 基于 jQuery Core 开发，提供了与 jQuery 一致的语法，保证了最小的学习曲线。
- ❑ 兼容所有主流的移动平台：iOS、Android、BlackBerry、Palm WebOS、Symbian、Windows Mobile、BaDa、MeeGo 以及所有支持 HTML 的移动平台。
- ❑ 轻量级（所有 mobile 的功能压缩后只有 12KB），以及最小图片依赖。
- ❑ HTML5 标记驱动：加快开发并且可以最小化需要运行的脚本。
- ❑ 渐进增强：jQuery Mobile 采用完全的渐进增强原则，通过一个全功能的 HTML 网页和额外的 JavaScript 功能层，提供顶级的在线体验。这意味着即使移动浏览器不支持 JavaScript，基于 jQueryMobile 的移动应用程序仍能正常地使用。
- ❑ 无障碍：包括 WAI-ARIA 在内的无障碍功能以确保页面能在类似于 VoiceOver 等语音辅助程序和其他辅助技术下正常使用。
- ❑ 简单的 API 为用户提供鼠标、触摸和光标焦点简单的输入法支持。
- ❑ 强大的主题化框架 jQuery Mobile 提供强大的主题化框架和 UI 接口。

## 15.2 jQuery Mobile 入门

本小节将讲解 jQuery Mobile 的使用。因为篇幅有限不会细致到每一个分类函数。但是通过本节可以领略到 jQuery Mobile 的独特魅力。

## 15.2.1 Hello Mobile 实例

在深入学习之前，首先通过一个最简单的“Hello Mobile”实例，来快速地上手 jQuery Mobile。

首先看一下运行效果，已进入页面，首先是一个拥有头部、中部、尾部三部分的页面，中间的 Say Hello 是一个可点击的 a 元素，效果如图 15-1 所示。

单击中间的“Say Hello”，会发现页面没有任何刷新，而变成如图 15-2 所示的内容。

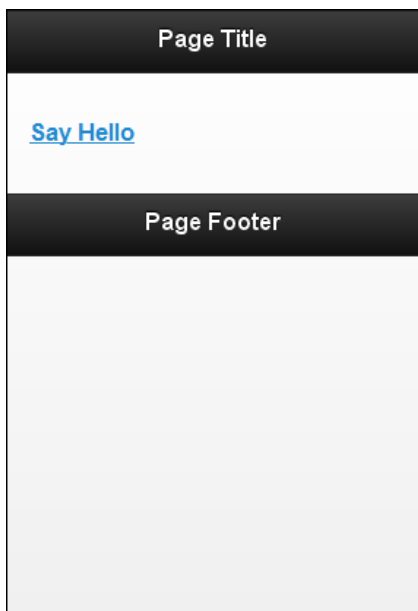


图 15-1 HelloMobile 实例首页

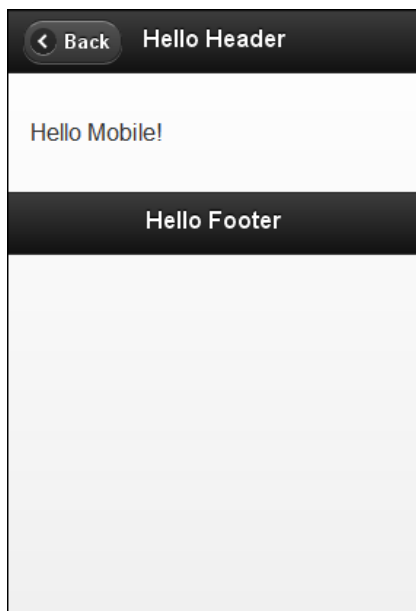


图 15-2 HelloMobile 实例单击页面

在图 15-2 所示的页面上，单击头部的“Back”按钮，可以回到首页。

本实例是一个最简单的 jQuery Mobile 应用，因为实现上面的效果，不需要编写任何的 javascript 代码，而只需要单纯地引入 jQuery Mobile 类库即可。下面是本实例的完整代码：

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello Mobile! </title>
  <link href="../Static/common/css/jquery.mobile-1.0a2.min.css"
rel="stylesheet" type="text/css" />
  <script src="../Static/common/js/jquery-1.4.4.min.js"
type="text/javascript"></script>
  <script src="../Static/common/js/jquery.mobile-1.0a2.js"
type="text/javascript"></script>
</head>
<body>
  <!-- 主页面 -->
  <div data-role="page">
    <div data-role="header">
      <h1>Page Title</h1>
    </div>
    <div data-role="content">
      <p><a href="#pageHello">Say Hello</a></p>
    </div>
    <div data-role="footer">
```



```

        <h4>Page Footer</h4>
      </div>
    </div>

    <!-- 弹出页 -->
    <div id="pageHello" data-role="page">
      <div data-role="header">
        <h1>Hello Header</h1>
      </div>
      <div data-role="content">
        <p>Hello Mobile!</p>
      </div>
      <div data-role="footer">
        <h4>Hello Footer</h4>
      </div>
    </div>
  </body>
</html>

```

这个示例首先引入了 jQuery Mobile 类库的 CSS 文件、jQuery 类库的 js 文件及 jQuery Mobile 类库的 js 文件。要使用 jQuery Mobile 及其皮肤，这 3 个文件都是必不可少的。

注意到这个例子没有任何的 javascript 代码，但是页面有两个结构化的 div 元素。其中：

```

<!-- 主页面 -->
<div data-role="page">

```

这个 div 中的内容就是实例首页。

另一个 div 元素：

```

<!-- 弹出页 -->
<div id="pageHello" data-role="page">

```

里面的内容就是弹出页。

超链接元素“Say Hello”是如何与这个弹出页关联的呢？只需要将“a”元素的 href 属性，设置为传统网页的“锚点”即可：

```

<a href="#pageHello">Say Hello</a>

```

此锚点的值是“#+弹出页 div 的 id 值”。

通过这个示例，可以领略到 jQuery Mobile 的一个特点：使用 div 组织页面。现在的页面已经不再是传统意义上的单个 html 文件。在 jQuery Mobile 的世界中，div 就是页面的容器。下面会详细讲解。

另外，之所以没有些任何的 javascript 语句，是因为在引用 jQuery Mobile 的 js 文件时，会对页面进行初始化操作。后面也会详细讲解。

## 15.2.2 API 分类

通过上面的快速入门，我们发现原来构造一个跨手机浏览器、并且样式美观的页面是如此的简单。

首先学习 jQuery Mobile 的函数分类（见表 15-2），为系统地学习 jQuery Mobile 和以后自行查询 API 打好基础。



表 15-2 jQuery Mobile API 分类

分类名称	分类中的函数作用
Ajax	Ajax 操作
Core	核心基础功能
CSS	CSS 相关
Data	数据相关
Dimensions	测量元素宽、高等
Effects	实现动画效果
Forms	表单操作
Manipulation	控制位置、包装元素以及控制元素宽高。Dimensions 分类中的部分函数也在此分类下。
Miscellaneous	杂项。无法归类的函数都在这里。
Offset	计算元素和鼠标位置
Plugin Authoring	插件相关功能
Properties	jQuery 对象集合的一些属性
Selectors	jQuery 选择器
Traversing	管理 jQuery 对象集合
Utilities	工具函数

jQuery Mobile 中的很多函数分类与 jQuery 类似,但是也有很多是 jQuery Mobile 本身所特有的。比如: 渐进增强、将 page 组织成 div 等。这些都是有别于传统的页面开发的。

随着 jQuery Mobile 的不断完善和改进,分类也有可能发生变化。由于篇幅原因本书不会对 jQuery Mobile 的每个函数都进行讲解。下面会介绍 jQuery Mobile 的主要功能和特点,学习了 jQuery Mobile 的特色后,如果需要某些功能,可以自行在 API 分类中查找。

### 15.2.3 页面结构

在“Hello Mobile”实例中,已经简单领略了 jQuery Mobile 独特的页面结构。

页面是构造一个网站的基础。jQuery Mobile 中的页面,除了传统意义上的一个独立的文件(HTML、PHP 等),还可以是内嵌在一个页面内部的“内部链接页”。也就是实例中的一个一个“div”。使用这种独特的接口,可以实现丰富的用户体验,实现传统 HTTP 请求所不能实现的效果。

使用 jQuery Mobile 的“页面”,首先需要保证页面头部引用了 HTML 标准的“DOCTYPE”:

```
<!DOCTYPE html>
```

引用了此标签才能保证 jQuery Mobile 的框架可以支持所有的功能。如果旧设备不支持此标签,也会被安全地忽略。并且在头部,需要引用 jQuery Mobile 的 CSS.js 文件以及 jQuery 的 js 文件。如果不想下载这些文件,可以直接引用 jQuery Mobile 官方的地址,下面是一个完整的头部:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
```

```
<link rel="stylesheet"
href="http://code.jquery.com/mobile/1.0a1/jquery.mobile-1.0a1.min.css" />
<script src="http://code.jquery.com/jquery-1.4.3.min.js"></script>
<script src="http://code.jquery.com/mobile/1.0a1/jquery.mobile-1.0a1.min.js">
</script>
</head>
```

在页面内所有的“页面”元素，即“div”元素，需要添加上“data-role”属性并设置值为“page”：

```
<div data-role="page">
...
</div>
```

在一个“page”内，可以存在头部、主体部分、尾部三部分，同样也是使用“data-role”属性标注：

```
<div data-role="page">
  <div data-role="header">...</div>
  <div data-role="content">...</div>
  <div data-role="footer">...</div>
</div>
```

标注了“data-role”属性的 div，都是容器。其中可以填写页面内容。

一个页面内可以防止多个“页面”，并且可以实现浏览器不刷新更换页面。使用 a 元素的 href 属性，可以实现两个页面容器的切换。下面的例子说明了如何切换内部链接和外部链接：

```
<div data-role="page">
  <div data-role="header">
    <h1>Linked Page</h1>
  </div>
  <div data-role="content">
    <p><a href="#pageInternal">Internal Page Link</a></p>
    <p><a href="External.htm" rel="external">External Page Link</a></p>
  </div>
</div>
```

在上面页面容器的内容部分，“Internal Page Link”表示内部链接。单击此链接会切换到页面上 id 为“pageInternal”的页面容器。因为 jQuery Mobile 默认将 a 元素的 href 属性设置为内部链接，所以如果想链接到外部的“External.htm”页面时，必须要添加上额外的“ref”属性：

```
<a href="External.htm" rel="external">
```

否则单击“External Page Link”时，将默认在页面内部，寻找 id 为“External.htm”的 div 页面容器。

两个页面切换时，会有默认的“slide”滑动动画效果。jQuery Mobile 有很多切换效果，可以通过 a 元素的“data-transition”属性设置。比如下面的例子使用“pop”弹出式效果切换页面：

```
<a href="#pageInternal" data-transition="pop">Page-Pop</a>
```

jQuery Mobile 支持的动画效果包括：

- ☐ slide: 滑动动画
- ☐ slideup: 向上滑动
- ☐ slidedown: 向下滑动





- ❑ pop: 弹出效果
- ❑ fade: 飞入效果
- ❑ flip: 跳出效果

除了改变页面的动画效果，还可以进一步改变页面的展现形式。可以将页面以“对话框”的形式展现。不同之处就是展现形式，“对话框”的展现形式如图 15-3 所示。

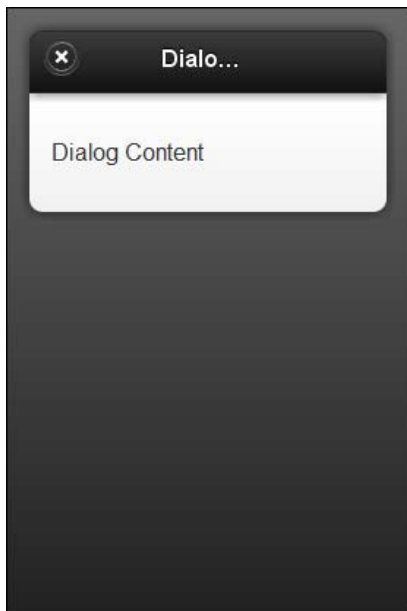


图 15-3 dialog 形式的页面

想让页面展现成 dialog 的形式，首先需要将页面容器的“data-role”属性设置为“dialog”，比如下面的例子：

```
<div id="pageDialog" data-role="dialog">
  <div data-role="header">
    <h3>Dialog Header</h3>
  </div>
  <div data-role="content">
    <p>Dialog Content</p>
  </div>
</div>
```

另外，还需要在链接到这个页面的连接上，也添加“data-role”属性：

```
<a href="#pageDialog" data-rel="dialog">Page-Dialog</a>
```

这就完成了一个“对话框”。

#### 15.2.4 配置系统

为什么只要按照特定的结构写 HTML，就能让 div 变成“页面”呢？这是因为当引用 jQuery Mobile 的 js 文件时，jQuery Mobile 会自动添加下面的事件：

```
$(document).bind("mobileinit", function(){...});
```

这个事件是在“document.ready”事件之前触发的，这是和 jQuery 最大的不同。而且因为是在引用脚本时就添加了此事件，所以如果我们希望重写此事件，修改 jQuery Mobile 的



一些默认行为和配置，就需要在 jQuery Mobile 的 js 文件引入前重写此事件。通常建议将脚本放在 jQuery Mobile 之前加载，比如：

```
<script src="jquery.js"></script>
<script src="custom-scripting.js"></script>
<script src="jquery-mobile.js"></script>
```

现在可以在“custom-scripting.js”文件中，添加用户自己的“mobileinit”事件：

```
$(document).bind("mobileinit", function(){
    $.extend( $.mobile , {
        foo: bar
    });
});
```

在“mobileinit”事件中，通过修改“\$.mobile”对象，可以保持 jQuery Mobile 的默认配置。比如页面变换的效果默认是“slide”，下面的代码会修改成“pop”：

```
<script type="text/javascript">
$(document).bind("mobileinit", function ()
{
    $.extend($.mobile, {
        defaultTransition: "pop"
    });
});
</script>
<script src="../../Static/common/js/jquery.mobile-1.0a2.js"
type="text/javascript"></script>
```

注意要在 jQuery Mobile 的 js 文件引入前，添加自己的“mobileinit”事件。

有关“\$.mobile”的都有哪些可以设置的配置项，请参考 jQuery Mobile 的 API 文档。

### 15.2.5 事件处理

jQuery Mobile 的事件处理和 jQuery 一样，都是用 bind 或者 live 函数进行事件绑定。但是手机浏览网页不同于电脑，大部分都是触摸屏，都需要各种触摸操作，比如拖动、滑动等等。另外手机的重力感应模块还可以根据手机是水平的还是垂直的，更改页面的显示行为是横向还是纵向。

jQuery Mobile 针对这些手机浏览器的特殊行为，添加了一些特殊的事件。虽然这些事件不是标准的，但是十分有帮助。表 15-3 是目前 jQuery Mobile 的一些特殊的用户事件。

表 15-3 jQuery Mobile 特殊用户事件

事件函数	相关说明
tap	一个快速完整的触屏事件
taphold	按住不放的触屏事件（按住接近一秒）
swipe	1 分钟内，水平移动超过 30px（并且垂直移动小于 20px）时触发
swipeleft	向左方向移动
swiperight	向右方向移动
orientationchange	当手机的浏览方向改变时（横、纵改变）触发
scrollstart	滚动条开始滚动时触发
scrollstop	滚动条结束滚动时触发
pagebeforeshow	在页面开始显示前触发



(续表)

事件函数	相关说明
pagebeforehide	在页面开始消失前触发
pageshow	在页面显示后触发
pagehide	在页面消失后触发
pagebeforecreate	在页面被初始化前触发
pagecreate	在页面被初始化以后触发

可以看到这些事件对于开发手机应用，增加手机用户体验都是十分有帮助的。

下面的例子可以在每一个“pageHello”页面显示后，弹出一个对话框，显示上一个页面的 id:

```
$("#pageHello").live("pageshow", function (event, ui)
{
    alert("This page was just show:" + ui.prevPage[0].id);
});
```

应用 jQuery Mobile 的这些事件，可以创造出手机特有的用户体验。

jQuery Mobile 还有许许多多特性比如皮肤系统、各种用户控件等。本书无法详细地讲解和剖析 jQuery Mobile，只能粗略地领略 jQuery Mobile 的魅力。下面通过一个简单完整的实例，来构建一个有具体功能的页面。

### 15.3 jQuery Mobile 与百度地图 API 综合实例

在这里又要再次请出百度地图 API。原因就是百度地图 API 是支持 Andriod 和 iPhone 系统的，可以使用百度地图 API 配合 jQuery Mobile，创建基于手机的地图应用。

#### 15.3.1 实例效果

首先来看最终的实现效果，实例首页如图 15-4 所示。在此页面的“footer”上放置了 3 个功能按钮，分别用来实现“本地搜索”、“公交”和“驾车”功能。单击按钮会在当前页面弹出录入控件。比如单击本地搜索后，页面弹出框如图 15-5 所示。



图 15-4 移动版百度地图实例首页



图 15-5 移动版百度地图实例弹出页

在此页面，如果只输入“搜索地点”，则是地点搜索，比如搜索“中关村”，点击搜索后会在地图上查找满足关键字“中关村”的地点，并且在地图上显示，效果如图 15-6 所示。如果同时输入了“搜索地点”与“在周围搜索”，则会在指定的地点周围查找如“小吃”等商户，效果如图 15-7 所示。



图 15-6 本地搜索效果



图 15-7 周边搜索效果

因为手机的窗口很小，所以无法同时显示文字结果内容。单击图 15-6 上面的标注点，可以查看具体的商户信息。

“公交”和“驾车”功能与上一章的例子相同。但是因为缺少“结果容器”，所以无法表述精确的文字信息。但是如果深度二次开发，这些功能都是可以实现的。就像百度地图官网自己的实现一样。

本实例可以在下面的代码中找到：

代码 15-1 移动版百度地图实例【jQueryStorm.Web/chapter15/MobileMap.htm】

下面就本实例的关键知识点进行讲解。

### 15.3.2 定制页脚

本实例将功能按键放在了页脚。下面是本实例主页面的 HTML 代码：

```
<!-- 主页面 -->
<div data-role="page">
  <div data-role="header">
    <h1>百度地图 API</h1>
  </div>
  <div data-role="content">
    <div id="mapContainer">
    </div>
  </div>
  <div data-role="footer" class="ui-bar">
    <a href="#pageLocal" data-rel="dialog" data-role="button">本地搜索</a>
```



```
<a href="#pageTransit" data-rel="dialog" data-role="button">公交</a>
<a href="#pageDriving" data-rel="dialog" data-role="button">驾车</a>
</div>
</div>
```

注意在页脚内，放置了 3 个“a”元素。为了让“a”元素可以显示成按钮的样子，为每个元素添加了“data-role”属性。这其实是应用了 jQuery Mobile 的 button 组件，将“a”元素显示成了按钮的效果。

“data-rel=dialog”是为了让内嵌页面显示成对话框，前面的章节中已经讲解过。

### 15.3.3 组织页面脚本

同第 14 章的实例一样，同样使用了面向对象的形式组织页面脚本。需要引用在第 13 章我们自己打造的脚本框架“core.js”文件，主要是使用其中的两个函数：

- ❑ proxy 函数：用于改变函数的“this”上下文。
- ❑ Class.create 函数：用于创建类实例。

另外还要引用百度地图 API 的脚本，所以本实例引用了很多的 js 文件：

```
<head>
<title>jQuery Mobile And Baidu Map</title>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<link href="../Static/common/css/jquery.mobile-1.0a2.min.css"
rel="stylesheet" type="text/css" />
<script src="../Static/common/js/jquery-1.4.4.min.js"
type="text/javascript"></script>
<script src="../Static/common/js/Core.js" type="text/javascript"></script>
<script src="../Static/common/js/jquery.mobile-1.0a2.js"
type="text/javascript"></script>
<script type="text/javascript"
src="http://api.map.baidu.com/api?key=cfc4eebecf9df9fb52c6a3eebc98e877&v=1.1&services=true"></script>
</head>
```

除了百度地图 API 的脚本，其他的脚本文件可以压缩合并成一个文件。

本页面首先使用 JSON 的方式创建了一个类“ThisPage”：

```
var ThisPage = { ... }
```

然后，在 document.ready 事件中，创建类实例：

```
$(function ()
{
    var thisPage = Class.create(ThisPage)
});
```

这些知识点都是前几章学习时强调过的。在移动网站的开发时同样可以使用。

### 15.3.4 添加事件

使用 jQuery Mobile，会发现本实例对于事件绑定的处理与第 14 章几乎完全一致。有关百度地图 API 的介绍，本章不再做介绍，可以通过第 14 章学习百度地图 API。下面说一下本实例不一样的地方。

因为手机分辨率是多种多样、不确定的。所以本实例要能够自动适应屏幕大小。所以 ThisPage 类上有一个“resizeMap”方法：

```
resizeMap: function ()
{
```



```
$("#mapContainer").height($(window).height() - 120);
$("#mapContainer").width($(window).width() - 20);
},
```

此方法会在脚本对象初始化时执行，即在页面加载时首先确定地图容器的大小：

```
initialize: function ()
{
    this.resizeMap();
    .....
}
```

另外在浏览器的 **resize** 事件中，也会调用此方法：

```
$(window).resize(this.resizeMap.proxy(this));
```

这样本实例不仅可以兼容所有分辨率的手机，而且在浏览器的大小改变时也会自动适应。

本实例的用户输入区域都放在了不同的“页面”div 中。同样可以使用 id 选择器和 bind 函数，为每一个搜索功能添加事件。比如本地搜索：

```
//本地搜索
$("#btnSearchFor").click(function (e)
{
    .....
}).proxy(this);
```

当单击“btnSearchFor”按钮时，会在地图容器上显示搜索结果。但是弹出框的隐藏以及地图的显示是哪里控制的呢？本实例应用了 jQuery Mobile 弹出框的默认行为。即单击其中的“button”控件（一个 a 元素）时，默认是关闭 dialog 弹出框回到主页上，从而实现了弹出框的切换。

借助 jQuery Mobile，仅仅做了很小的改动，就将第 14 章的实例，移植到了手机上。这都得益于：首先是 jQuery 和 jQuery Mobile 两个类库的相似性，再就是百度地图 API 可以支持手机应用。

## 15.4 小结

本章介绍了 jQuery Mobile，并且简单介绍了如何使用 jQuery Mobile 开发手机网站。但是由于篇幅有限无法详细介绍 jQuery Mobile 的方方面面。但是可以从实例中，感受到 jQuery Mobile 的便利和强大。

jQuery Mobile 拥有一个宏伟的目标，作为第一个手机脚本库，jQuery Mobile 还具有很多的潜力。如果想进入移动开发，jQuery Mobile 是一个非常好的选择。

